# Mining Data Streams II

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University
May 19, 2022

# TODAY

*Mining Data Streams II: Overview*

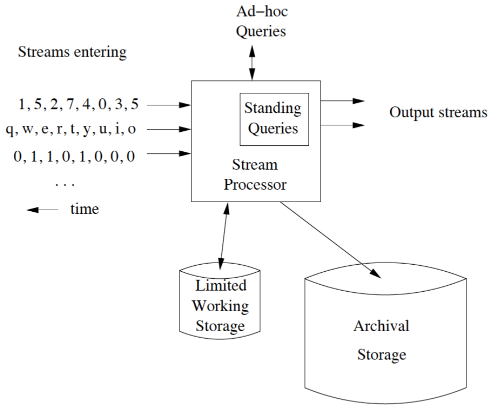
- Counting Ones in a Window:
  ☞ Datar-Gionis-Indyk-Motwani algorithm
- Decaying Windows

*Learning Goals:* Understand these topics and get familiarized

*Counting Ones in a Window*

*The Datar-Gionis-Indyk-Motwani Algorithm*

# Data Stream Management System




A data stream management system

Adopted from `mmds.org`

# DATA STREAM QUERIES

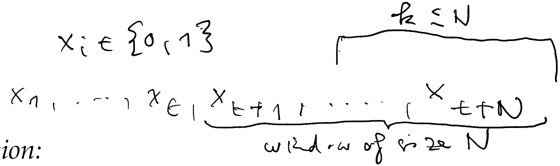
Issues

- Streams deliver elements rapidly: need to act quickly
- Thus, data to work on should fit in main memory
- New techniques required:

☞ Compute approximate, not exact answers

☞ Hashing is a useful technique

# Counting Ones in Window: Problem




- *Situation:*
  - Suppose we have a window of length $N$ on a binary stream
  - Query: "how many ones are there in the last $k \leq N$ bits?"
  - We cannot afford to store entire window
  - Approximate algorithms required
- Present solution for binary streams first
- Discuss extension for summing numbers (from a stream of numbers) thereafter

# The Cost of Exact Counts

- One needs to store $N$ bits to answer count-one-queries for arbitrary $k \leq N$:
  - Assume one could use less than $N$ bits
  - We need $2^N$ different representations to represent all possible $2^N$ bit strings of length $N$
  - Since we use less than $N$ bits, there are two different bit strings $w \neq x$, for which we use the same representation
  - Let $k$ be the first bit from the right where $w$ and $x$ disagree
  - *Example:*
    - For $w = 0101, x = 1010$, we have $k = 1$
    - For $w = 1001, x = 0101$, we have $k = 3$
  - So the counts of ones in the window of length $k$ for $w$ and $x$ differ
  - But because we use identical representations for $w$ and $x$, we will output the same count
  - This proves that one needs the full $N$ bits to represent bit strings for exact count-one-queries.

# THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM

$x_1, \ldots, x_t, x_{t+1}, \ldots, x_M$ (timestamp)

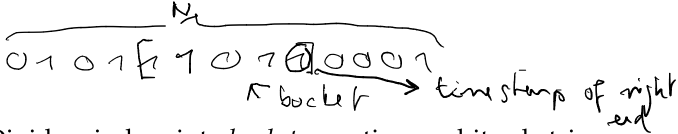
- *Situation:*
  - We consider a binary stream: elements are *bits*
  - Let each element of the stream have a *timestamp*
  - The first, *leftmost* element has timestamp 1, the second leftmost has timestamp 2, and so on
- *Goal:* We like to count the ones among the $N$ most recent (rightmost) elements/bits

  $N = 10^6$
- *Space requirements:*

  $[1000, 1001, \ldots, 10^6, 1, \ldots$ ($10^6 = 0$)
  - Storing timestamps modulo $N$, and
  - marking rightmost timestamp as most recent
  - allows to store positions of individual bits using $\log_2 N$ bits

$x_1, \ldots, x_t, \ldots, x_e, [x_{e+1}, \ldots, x_{e+N}]$

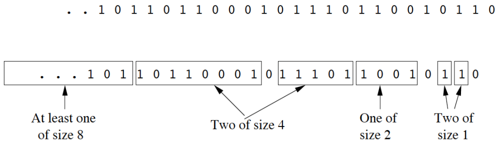
# The Datar-Gionis-Indyk-Motwani Algorithm




- *Algorithm:* Divide window into *buckets*, contiguous bit substrings
- *Bucket Representation:* For identifying buckets, we store
  - The timestamp of its right end, and
  - The *size* of the bucket, as the number of 1's in the bucket
  - The size is supposed to be a power of 2
- *Bucket Space Requirements:*
  - Timestamp requires $\log_2 N$ bits
  - Size is $2^j$, hence requires $\log\log_2 N$ bits (by storing $\log_2 j$ bits)
  - Requires $O(\log N)$ bits overall

storing bucket : [timestamp, $2^j$] where $2^j \leq N$, $j \leq \log_2 N$ → $\log \log N$

# DATAR-GIONIS-INDYK-MOTWANI RULES




Bit stream divided into buckets following DGIM rules

From mmds.org

- Right end always is a 1
- Every 1 of the window is in some bucket
- Buckets do not overlap
- All sizes must be a power of 2
- For each possible size, there are either one or two buckets
- Size of buckets cannot decrease when moving

# The Datar-Gionis-Indyk-Motwani Algorithm

*Key Ideas / Considerations*

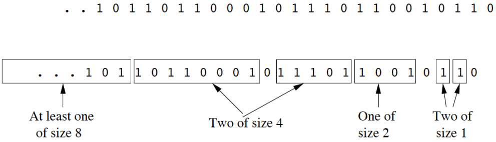
- The number of buckets representing a window must be small
- Estimate the number of 1's in the last $k$ bits (for any $k$) with an error of no more than 50%
- How to maintain the DGIM Bucket Rules on new bits arriving?

# The Datar-Gionis-Indyk-Motwani Algorithm

*Storage Requirements*

- Each bucket can be represented using $O(\log N)$ bits
- Let $2^j$ be size of largest bucket: $2^j < N$ implies $j \leq \log_2 N$
- So there are at most 2 buckets of sizes $2^j, j = \log_2 N, ..., 1$
- This means that there are $O(\log N)$ buckets
- Each bucket being represented by $O(\log N)$ bits requires $O(\log^2 N)$ space overall

# THE DATAR-GIONIS-INDYK-MOTWANI ALGORITHM



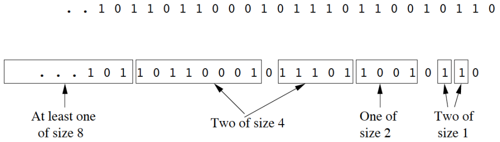

Bit stream divided into buckets following DGIM rules

From `mmds.org`

*Answering Queries*

- Let $1 \leq k \leq N$: how many 1's are among the last $k$ bits?
- *Answer:*
  - Find leftmost (= with earliest timestamp) bucket $b$ containing some of last $k$ bits
  - *Estimate:* Sum of sizes of buckets right of $b$ plus half the size of $b$

# The Datar-Gionis-Indyk-Motwani Algorithm




Bit stream divided into buckets following DGIM rules

From `mmds.org`

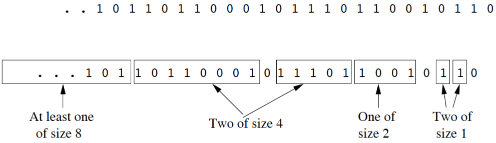
*Example*

- Let $k = 10$: how many 1's are among 0110010110?
- Let $t$ be timestamp of rightmost bit
- Two buckets with one 1 each, having timestamps $t-1$, $t-2$ are fully included in $k$ righmost bits
- Bucket of size 2 with timestamp $t-4$ is also included
- Bucket of size 4 with timestamp $t-8$ is only partially included
- Estimate: $1 + 1 + 2 + (1/2 \times 4) = 6$, one more than true count

# DGIM: Error of Estimate

estimate $\frac{c - 2^{j-1}}{c} = 1 - \frac{2^{j-1}}{c} \geq 0.5$

$\leq 0.5$




Bit stream divided into buckets following DGIM rules

From mmds.org

[1 0 1 1 0 1 1 0 1 1 0 1] [ . . . .

k

belongs to true count

8 for true count

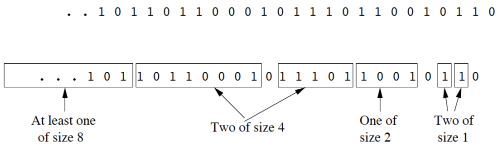
4 for estimate

*Case 1: estimate is less than c*

- Let $c$ be true count; let leftmost bucket $b$ be of size $2^j$
- *Worst case:* all 1's in $b$ are among $k$ most recent bits
- So, estimate is lower by $1/2 \times 2^j = 2^{j-1}$ than $c$
- Because $c \geq 2^j$, error is at most half of $c$

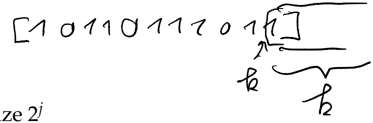
estimate

# DGIM: Error of Estimate




Bit stream divided into buckets following DGIM rules

From `mmds.org`



*Case 2: estimate is larger than c*

- Let $c$ be true count; let leftmost bucket $b$ be of size $2^j$
- *Worst case:* only rightmost bit of $b$ is among $k$ most recent bits, and
- There is only one bucket for each of sizes $2^{j-1}, \ldots, 1$
- That yields $c = 1 + 2^{j-1} + \ldots + 1 = 1 + 2^j - 1 = 2^j$
- Estimate is $2^{j-1} + 2^{j-1} + \ldots + 1 = 2^{j-1} + 2^j - 1$, so
- Error $\frac{2^{j-1}+2^j-1}{2^j}$ is no greater than 50% of true count

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

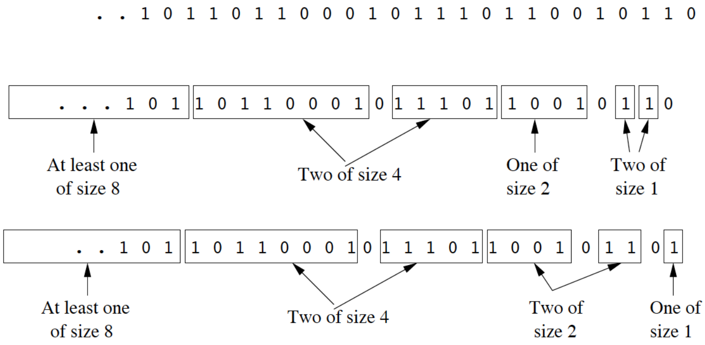
# Maintaining DGIM Rules

Upon a new bit with timestamp $t$ having arrived:

- Check timestamp $s$ of leftmost bucket $b$:
  - if $s \leq t - N$, drop $b$ from list of buckets
- If the new bit is 0, do nothing
- If the new bit is 1, do
  - Create new bucket with timestamp $t$ and size 1
  - On increasing size, starting with size 1, while there are three buckets of the same size, do
    - keep the rightmost bucket of that size as is
    - join the two left buckets into one of double the size
    - where the timestamp is that of the rightmost bit
  - *For example:* joining the two left of the three buckets of size 1 into a bucket of size 2 may create a third bucket of size 2, and so on
- *Runtime:* Need to look at $O(\log N)$ buckets, joining is constant time, so processing new bit requires $O(\log N)$ time overall

# The Datar-Gionis-Indyk-Motwani Algorithm

## Part VI




Bit stream divided into buckets following DGIM rules (top), with new 1 arriving (bottom)

From `mmds.org`

# DGIM Algorithm: Reducing the Error

- For some $r > 2$, allow either $r$ or $r - 1$ buckets of the same size
- Allow this for all but size 1 and largest size, whose numbers may be any of $1, ..., r$
- Compute estimate as before
- Extend maintaining the DGIM Bucket Rules in the obvious way
- *Recall:* largest error $\frac{2^{j-1}+2^j-1}{2^j}$ was made when only one 1 from leftmost bucket $b$ was within window
- *New error:*
  - True count is at most $1 + (r-1)(2^{j-1} + ... + 1) = 1 + (r-1)(2^j - 1)$
  - Estimate is $2^{j-1} + (r-1)(2^j - 1)$, difference between estimate and true count is $2^{j-1} - 1$, so fractional error is

$$\frac{2^{j-1} - 1}{1 + (r-1)(2^j - 1)}$$

    which is upper bounded by $1/2(r-1)$
  - Picking large $r$ can limit error to any $\epsilon > 0$

# DGIM ALGORITHM: EXTENSIONS

- DGIM can be extended to integers instead of bits
- Question is to estimate the sum of last $k \leq N$ integers from a window of $N$ integers overall
- However, DGIM cannot be extended to streams containing negative integers
- Consider case of integers in range of 1 to $2^m$, so represented by $m$ bits
- *Solution:*
  - Treat each bit of integers as separate stream
  - Apply DGIM algorithm to each of $m$ streams, yielding estimate $c_i$ for $i$-th stream
  - *Overall estimate:*

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

  - If error is at most $\epsilon$ for all $i$, overall error is also at most $\epsilon$

*Most Common Elements*

*Decaying Windows*

# Decaying Windows: Motivation

- *Stream:* Movie tickets purchased all over the world
- *Goal:* Listing currently most “popular” movies
- *Currently popular:*
  - Movie that sold plenty of tickets years ago not to be listed
  - Movie that sold $2n$ tickets last week, for large $n$, currently popular
  - Movie that sold $n$ tickets in last 10 weeks is even more popular
  - How to grasp that idea?

# Decaying Windows: Motivation

- *Stream:* Movie tickets purchased all over the world
- *Goal:* Listing currently most "popular" movies
- *Possible solution:*
  - One bit stream for each movie
  - The i-th bit in a movie stream is 1 if the i-th ticket was for that movie
  - Pick window of size $N$, where $N$ is to reflect tickets to be recent
  - Estimate number of ones in each stream
    - Use Datar-Gionis-Indyk-Motwani (DGIM) algorithm, for example
    - Estimates number of tickets sold for each movie
  - Rank movies by the estimated counts

M1: 0 0 0 1 ....
M2: 1 0 0 0 ....
M3: 0 1 1 0 ....

# DECAYING WINDOWS: MOTIVATION

- *Possible solution, summary:*
    - One bit stream for each movie
    - i-th bit in a movie stream is 1 iff i-th ticket was for that movie
    - Count number of ones in each stream...
    - ... counts tickets for each movie
    - Rank movies by ticket counts
- Works for movies, because there only thousands of movies
- *Drawback:*
    - Does not work for items at Amazon or tweets per Twitter-user
    - ☞ too many items or users

# DECAYING WINDOWS: MOTIVATION

- *Stream:* Movie tickets purchased all over the world
- *Goal:* Listing currently most "popular" movies
- *Alternative approach:*
  - Do not count ones in fixed-size window
  - Rather, compute "smooth aggregation" of *all* ones in stream
  - Smooth: use weights to rate stream elements in terms of recentness
  - The further back in the stream, the less weight given

$x_1, \ldots, x_t, \ldots, x_M$

$w_1, \ldots, w_t, \ldots, w_M$ $\quad w_1 \leq w_2 \leq \ldots \leq w_t \leq \ldots \leq w_M$

# Exponentially Decaying Window: Definition

Definition [Exponentially Decaying Window]:

- Let $a_1, a_2, ..., a_t$ be a stream, with $a_t$ most recent element
- Let $c$ be small constant, e.g. $c \in [10^{-9}, 10^{-6}]$

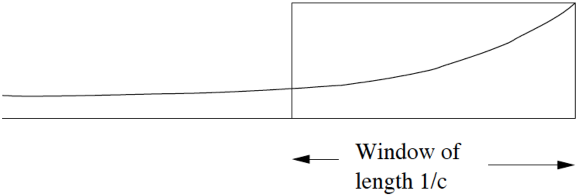
The *exponentially decaying window* for the stream is defined to be the sum

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^i \quad (1)$$

weight is $(1-c)^i$

the further to the left, the greater $i$

# Exponentially Decaying Window: Definition




Decaying window and fixed-length window of equal weight

From mmds.org

- Decaying window puts weight $(1-c)^i$ on $(t-i)$-th element
- A window of length $1/c$ puts equal weight 1 on the first $1/c$ elements
- Both principles distribute the same weight to stream elements overall

# Updating Exponentially Decaying Windows

Upon arrival of a new element $a_{t+1}$, one updates the exponentially decaying window $\sum_{i=0}^{t-1} a_{t-i}(1-c)^i$ by

1. multiplying the current window by $(1-c)$, yielding

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^{i+1}$$

2. adding $a_{t+1}$, yielding

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^{i+1} + a_{t+1} = \sum_{i=0}^{(t+1)-1} a_{(t+1)-i}(1-c)^i$$

# Exponentially Decaying Windows: Finding Most Popular Movies

- *Most Popular Movies: Idea*
  - Have a bit stream for each movie, as before
  - Use e.g. $c = 10^{-9}$ ($\approx$ sliding window of size $1/c = 10^9$)
  - On incoming movie ticket sale, update all decaying windows, as described above
    - First, multiply all decaying windows by $1 - c$
    - Add 1 for stream of the movie of the ticket; if there is no stream for that movie, create one
    - Do nothing (add 0) for all other streams
  - If any decaying window drops below threshold of $1/2$, drop window
  - Because the sum of all scores is $1/c$, there cannot be more than $2/c$ movies with score of $1/2$ or more
  - So, $2/c$ is limit on number of movies being tracked at any time
  - In practice, there should be much less movies counted
- *Therefore,* one can apply the technique also for Amazon items and Twitter-users

# Materials / Outlook

- See *Mining of Massive Datasets*, chapter 4.6, 4.7
- As usual, see `http://www.mmds.org/` in general for further resources
- Next lecture: “Link Analysis I”
  - See *Mining of Massive Datasets* 5.1–5.5