Stream Statistics Over Sliding Window

Anil Maheshwari

School of Computer Science
Carleton University
Canada

Outline

Introduction

Algorithm

Sum Problem

Trends

References

Introduction

Problem Setting

Main Problem

The input is an endless stream of binary bits. At any time, among the last N bits received, we are interested in queries that seek an approximate count of the number of 1's in the stream among the last k bits, where $k \leq N$.

Result: A data structure of size $O(\frac{1}{\epsilon}\log^2 N)$ that can approximate the count of the number of 1s within a factor of $1 \pm \epsilon$

Reference: Maintaining stream statistics over sliding windows by Datar, Gionis, Indyk, and Motwani, SIAM Jl. Computing 2002

Variants

1. A stream of positive numbers. The query consists of a value $k \in \{1, \ldots, N\}$, and we want to know the (approximate) sum of the last k numbers in the stream. (Uses sublinear space.)
2. A stream consisting of numbers from the set $\{-1, 0, +1\}$. We want to maintain the sum of last N numbers of the stream. (Requires $\Omega(N)$ bits of storage to approximate the sum that is within a constant factor of the exact sum.)
3. What are the most popular movies in the last week?
4. What is trending in the last week?
5. $\ldots$

Main Problem

Main Problem

Report an approximate count of the number of 1's in the stream of binary bits among the last k bits, where $k \leq N$.

What about Exact Count?

Algorithm

Algorithm for Approximate Count

Algorithm uses two structures:

Time Stamps: To track the most recent N bits.

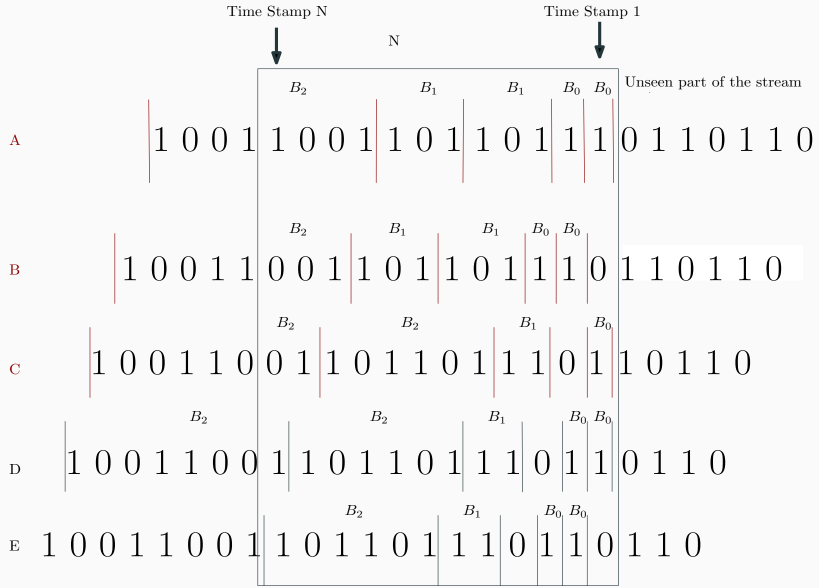
Buckets: With the following features:

- $O(\log N)$ buckets maintain the 1's among the latest N bits
- Number of 1's in a bucket is a power of 2
- Each 1-bit is assigned to exactly one bucket (0-bit may or may not be assigned to any bucket)
- At most two buckets of a given *size* (size $= \#1$s)
- Each bucket stores time stamp of its most recent bit
- Most recent bit of any bucket is 1-bit

Update

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

An Illustration



Algorithm contd.

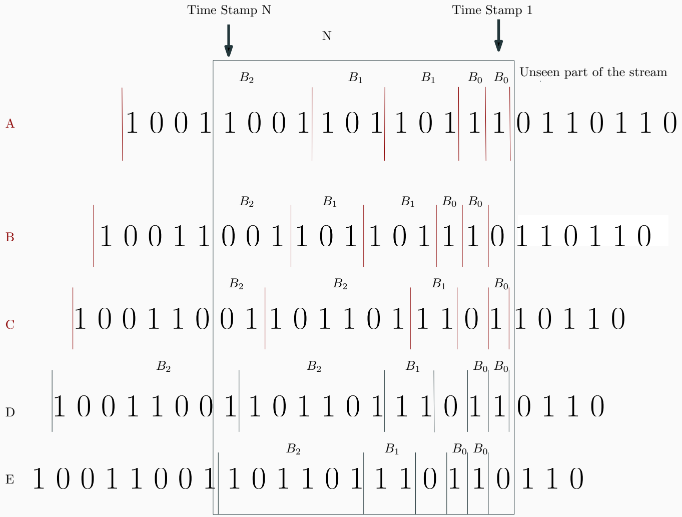
On receiving a new bit in the data stream:

0**-bit**: Increment the time stamp of each of the buckets by 1, and if any of the buckets time stamp exceeds N, we discard that bucket.

1**-bit**: Following updates are done:

1. Create a bucket B_0 consisting of the newest 1-bit with a time stamp of 1.
2. Scan the list of buckets in order of increasing size.
   Case 1: Two buckets of size B_0.
   Increment time stamp of each bucket (and possibly discard buckets whose time stamps exceed N)
   Case 2: Three buckets of type B_0.
   - Merge the two oldest buckets of type B_0 to form a bucket of type B_1.
   - Update time stamps appropriately of each bucket as in Case 1.
   - Now, if we have three buckets of type B_1, merge the oldest two B_1 buckets, to form a bucket of type B_2, and repeat.

Illustration



Space Complexity

We have:

- $O(\log N)$ buckets as the size of window is N
- Bucket B_i stores 2^i 1-bits
- For each bucket we store its time stamp and its size
- Time stamps requires $O(\log N)$ bits
- Storing i with bucket B_i is sufficient for its size
- As $0 \leq i \leq \log N$, i can represented using $O(\log \log N)$ bits
- Total space required
  $O(\log N(\log N + \log \log N)) = O(\log^2 N)$ bits

Time Complexity

On receiving a 0-bit:
- We update time stamps of each of the $O(\log N)$ buckets
- Requires $O(\log N)$ time

On receiving a 1-bit:
- We update the time stamps of each bucket
- Potentially, perform cascaded merging of buckets
- Time (merge & cascade) $\approx$ # of buckets
- Requires $O(\log N)$ time

Total Time (per update): $O(\log n)$

Answering Query

Query Problem

For any query value $k \in \{1, \ldots, N\}$, report an approximate count of the number of 1's among the latest k bits of the stream.

1. Initialize $C := 0$
2. Traverse buckets from right to left. For each bucket of type B_i that is encountered in the traversal:
    2.1 B_i is completely contained in the window:
        $C := C + 2^i$
    2.2 B_i is completely outside the window:
        C remains unchanged
    2.3 Partially overlaps the window:
        $C := C + \frac{2^i}{2}$
3. Report C as an approximate count

Analysis of Approximation Factor

2-factor approximation

Let C^* be the true count of number of 1s in the query window of size k. Then, $\frac{1}{2} \leq \frac{C}{C^*} \leq 2$.

Proof: Let B_l be the last bucket type that partially overlaps the query window.
- Anywhere from $2^0 = 1$ to 2^l 1's may contribute to the count C^*, whereas for C we take the contribution of this bucket to be $\frac{2^l}{2} = 2^{l-1}$.
- Thus we may incur an error.
- Note that all the previous buckets contribution is correctly accounted in the sum C.
- Let that contribution be α, where $\alpha \geq \sum_{i=0}^{l-1} 2^i = 2^l - 1$.
- Thus, $C = \alpha + 2^{l-1}$, $C^* \geq \alpha + 1$, and $\alpha \geq 2^l - 1$.
- We have $\frac{C}{C^*} \leq \frac{\alpha + 2^{l-1}}{\alpha + 1} \leq 1 + \frac{1}{2}$.
- Similarly, $C^* \leq \alpha + 2^l$, and thus $\frac{C}{C^*} \geq \frac{\alpha + 2^{l-1}}{\alpha + 2^l} \geq 1 - \frac{1}{2}$.

□

Improvements

Let $r > 2$ be an integer parameter.

Maintain $r - 1$ or r copies of B_i for each $i \geq 1$
Note: B_0 and the largest bucket may have fewer than $r - 1$ copies.

At any time, if we exceed r copies of any type of buckets, we take the oldest two buckets and merge them to form a new bucket of the next size.

Answer queries as before.

Improvements (contd.)

Claim

For this setting, we have $1 - \frac{1}{r-1} \leq \frac{C}{C^*} \leq 1 + \frac{1}{r-1}$.
If $r = 1 + \frac{1}{\epsilon}$, we obtain a data structure of size $O(\frac{1}{\epsilon} \log^2 N)$ that approximates the count of the number of 1s within a factor of $1 \pm \epsilon$.

Assume that B_l is the last bucket that overlaps the window.

Observe that the true Count $C^* \geq 1 + (r-1) \sum_{i=1}^{l-1} 2^i$.

Error between C and C^* is at most $\leq 2^{l-1} - 1$.
Therefore, $\frac{2^{l-1}-1}{1+(r-1)\sum_{i=1}^{l-1} 2^i} \leq \frac{1}{r-1}$.

Hence, $1 - \frac{1}{r-1} \leq \frac{C}{C^*} \leq 1 + \frac{1}{r-1}$.

Sum Problem

Computation of Sum

The Sum Problem

A stream of positive numbers. The query consists of a value $k \in \{1, \ldots, N\}$, and we want to know the (approximate) sum of the last k numbers in the stream.

5 7 2 3 9 4 1 6 11 2 4 3

Approach I: Computation of Sum

If the next number in the stream is x, insert x $1'$s in the stream

5 7 2 3 9 4 1 6 11 2 4 3

Approach II: Computation of Sum

Assuming d-bit numbers. For each bit position i, maintain a stream. Let C_i be the value of approximate number of $1's$ in the stream i. Report approximate sum as $\sum_{i=0}^{d-1} 2^i C_i$

	5	7	2	3	9	4	1	6	11	2	4	3
2^3	0	0	0	0	1	0	0	0	1	0	0	0
2^2	1	1	0	0	0	1	0	1	0	0	1	0
2^1	0	1	1	1	0	0	0	1	1	1	0	1
2^0	1	1	0	1	1	0	1	0	1	0	0	1

Trends

What is Trending?

Among the last 10^{12} movie tickets sold, list all *popular* movies?

Let $c := 10^{-3}$. Maintain (decaying) scores for movies whose threshold is at least $\tau \in (0, 1)$. For each new sale of ticket (say for Movie M):

1. For each movie whose score is being maintained, its new score is reduced by a factor of $(1 - c)$
2. If we have the score of M, add 1 to that score. Otherwise, create a new score for M and initialize it to 1
3. Remove any score that falls below τ

Questions

1. What is sum of all scores at any point of time?
   Show that the sum total of all the scores is $\frac{1}{c}$

2. How many scores are maintained at any given time?
   Assume $\tau = \frac{1}{2}$.
   Show that at most $\frac{2}{c}$ movies are maintained, each with a score $\geq \tau$.

3. What are changes if $\tau = \frac{1}{3}$.

Variants

1. Min/Max
2. Stream with $\pm$ numbers
3. Lower Bounds: Results are more-or-less optimal up to constant factors
4. . . .

References

Conclusions

Main References:

1. Maintaining stream statistics over sliding windows, by Datar, Gionis, Indyk, and Motwani, SIAM Jl. Computing 2002.
2. Chapter in MMDS book (mmds.org)
3. Chapter on Data Streams in My Notes on Topics in Algorithm Design