SEEM5020 Algorithms for Big Data
Streaming Algorithms (II)

Sibo WANG

Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
Definition 1 (Data Stream Model)

The data streaming model involves processing a finite sequence of $n$ integers drawn from a finite domain of size $m$. However, unlike traditional datasets, this sequence is not readily available for random access. Instead, the data arrives incrementally in the form of a continuous 'stream,' with each integer being presented one at a time.

Main challenges:

- In the data streaming model, accessing the input sequence typically allows for only a small number of passes, most likely just once.
- The streaming algorithms are restricted to use a space that is logarithmic or polylogarithmic in $m$ and $n$. 
Table of Contents

1 Finding Frequent Items in Data Streams
   • Counter-based Algorithms
   • Sketch-based Algorithms

2 Querying Over a Slide Window in Streams
   • The DGIM Algorithm
Misra-Gries Algorithm: A deterministic algorithm that has
- a space cost of $O\left(\frac{1}{\gamma} \log n \right)$.
- $(\gamma \cdot n, 1)$-approximation (or simply $\gamma \cdot n$-approximation when the probability is 1) guarantee: $f(e) - \gamma \cdot n \leq \hat{f}(e) \leq f(e)$, where $\hat{f}(e)$ is an estimate of the frequency $f(e)$ of element $e$.

Algorithm 1 (Misra-Gries Algorithm)

Let $S$ be a set of pairs in the form $\langle k, c \rangle$, initially empty;
for each element $e$ in the stream do
    if there exists a pair $\langle e, c \rangle \in S$ then
        update the pair as $\langle e, c + 1 \rangle$
    else if $|S| < \lceil 1/\gamma \rceil - 1$ then
        Add a pair $\langle e, 1 \rangle$ to $S$;
    else
        for each pair $\langle e, c(e) \rangle \in S$ do
            decrement $c(e)$ by 1 and remove the pair from $S$ if $c(e) = 0$
        end
end
How to estimate?

- If a pair $\langle e, c(e) \rangle \in S$, return $\hat{f}(e) = c(e)$;
- Otherwise, return 0.

Example 1

Consider a stream of elements $B, A, B, C, B, D, A, D, D, E, E, E, E$. Assume that $\gamma = \frac{1}{4}$. 
How to estimate?

- If a pair \( \langle e, c(e) \rangle \in S \), return \( \hat{f}(e) = c(e) \);
- Otherwise, return 0.

**Example 1**

Consider a stream of elements \( B, A, B, C, B, D, A, D, D, E, E, E, E \). Assume that \( \gamma = \frac{1}{4} \). Initially, we add \( \langle B, 1 \rangle \) to \( S \). When the next element \( A \) comes, \( S = \{ \langle B, 1 \rangle, \langle A, 1 \rangle \} \). For the next element \( B \), we update the pair for \( B \) as \( \langle B, 2 \rangle \). For the next element \( C \), we update \( S = \{ \langle B, 2 \rangle, \langle A, 1 \rangle, \langle C, 1 \rangle \} \).

When the next \( B \) comes, we update the pair for \( B \) as \( \langle B, 3 \rangle \). Next, when \( D \) comes, \( S \) already includes 3 pairs, we decrement the counter for each element and remove those with counter zero. Thus \( S = \{ \langle B, 2 \rangle \} \). We repeat this process, and finally \( S = \{ \langle B, 1 \rangle, \langle D, 1 \rangle, \langle E, 3 \rangle \} \).
Analysis of Misra-Gries Algorithm

We prove that the Misra-Gries algorithm provides $\gamma \cdot n$-approximation.

Proof.

We count the number of key-decrease operations on all elements in $S$. It happens only when the set $S$ is full. A key-decrease operation will decrement the count by 1 to all $\lceil 1/\gamma \rceil$ elements in $S$. Since there are $n$ elements, the counter for any element $e$ is decremented by at most

$$\frac{n}{\lfloor 1/\gamma \rfloor} \leq \frac{n}{1/\gamma} = \gamma \cdot n.$$ 

Thus, for any element $e$,

$$\hat{f}(e) = c(e) \geq f(e) - \gamma \cdot n.$$ 

Also, it is obvious that $f(\hat{e}) \leq f(e)$. Proof done.
Space-Saving Algorithm [2]

The key difference between Space-Saving and the Misra-Gries Algorithm:

- When a new element $e$ comes and the set $S$ is full, it kicks out the pair $\langle e_{\text{min}}, c_{\text{min}} \rangle$ with the min count, and insert a pair $\langle e, c_{\text{min}} + 1 \rangle$.

Algorithm 2 (Space-Saving Algorithm)

Let $S$ be a set of pairs in the form $\langle k, c \rangle$, initially empty;

for each element $e$ in the stream do

  if there exists a pair $\langle e, c \rangle \in S$ then
      update the pair as $\langle e, c + 1 \rangle$
  else if $|S| < \lceil 1/\gamma \rceil$ then
      Add a pair $\langle e, 1 \rangle$ to $S$
  else
      Let $\langle e_{\text{min}}, c_{\text{min}} \rangle$ be the pair with the minimum count in $S$;
      Create a new pair $\langle e, c_{\text{min}} + 1 \rangle$;
      Delete $\langle e_{\text{min}}, c_{\text{min}} \rangle$ from $S$ and add $\langle e, c_{\text{min}} + 1 \rangle$ to $S$;

end
Space-Saving Algorithm (Cont.)

Estimation of an element $e$: the same as Misra-Gries Algorithm:

- If a pair $\langle e, c(e) \rangle \in S$, return $\hat{f}(e) = c(e)$;
- Otherwise, return 0.

**Example 2**

Consider a stream of elements $B, A, B, C, B, D, A, D, D, E, E, E, E$. Assume that $\gamma = \frac{1}{3}$.
Space-Saving Algorithm (Cont.)

Estimation of an element $e$: the same as Misra-Gries Algorithm:
- If a pair $\langle e, c(e) \rangle \in S$, return $\hat{f}(e) = c(e)$;
- Otherwise, return 0.

Example 2
Consider a stream of elements $B, A, B, C, B, D, A, D, D, E, E, E$. Assume that $\gamma = \frac{1}{3}$. Initially, we add $\langle B, 1 \rangle$ to $S$. When $A$ comes, $S = \{\langle B, 1 \rangle, \langle A, 1 \rangle\}$. For the next element $B$, we update the pair for $B$ as $\langle B, 2 \rangle$. For the next element $C$, we update $S = \{\langle B, 2 \rangle, \langle A, 1 \rangle, \langle C, 1 \rangle\}$. When the next $B$ comes, we update the pair for $B$ as $\langle B, 3 \rangle$. Next, when $D$ comes, $S$ already includes 3 pairs, we choose a pair with the smallest count. Assume that we choose $\langle A, 1 \rangle$. We add a pair $\langle D, 2 \rangle$ and remove $\langle A, 1 \rangle$ from $S$. Thus $S = \{\langle B, 2 \rangle, \langle C, 1 \rangle, \langle D, 2 \rangle\}$. When element $A$ comes, we repeat the same steps and update $S$ as $S = \{\langle B, 3 \rangle, \langle D, 2 \rangle, \langle A, 2 \rangle\}$. When the stream ends, $S = \{\langle B, 3 \rangle, \langle D, 4 \rangle, \langle E, 6 \rangle\}$. 
The Space-Saving algorithm has the following properties.

**Lemma 1 (Overestimation of Maintained elements)**

For an element \( e \) that is in set \( S \), its estimation \( \hat{f}(e) = c(e) \) is never under-estimated.

**Lemma 2 (Upper bound of \( c_{min} \))**

Let \( c_{min} \) be the minimum count value in \( S \), \( c_{min} \leq \gamma \cdot n \).

**Lemma 3 (Properties of elements in \( S \) and not in \( S \))**

If \( f(e) > c_{min} \), the element \( e \) should appear in the set \( S \). For any element \( e \) that does not appear in \( S \), \( f(e) \leq \gamma \cdot n \).

**Lemma 4 (\( \gamma \cdot n \)-approximation of Space-Saving)**

For any element \( e \), Space-Saving provides \( \gamma \cdot n \)-approximation.
1 Finding Frequent Items in Data Streams
   - Counter-based Algorithms
   - Sketch-based Algorithms

2 Querying Over a Slide Window in Streams
   - The DGIM Algorithm
The Count-Min sketch has the following structure:

- It maintains $d$ arrays, each with a size of $w$. For each array $A[i][1 \cdots w]$, a hash function $h_i : [m] \rightarrow [w]$ is drawn from 2-wise independent family $\mathcal{H}$ and is associated with this array $A[i][1 \cdots w]$;
- When $e_j$ comes, for each array $A[i]$, update $A[i][h(e_j)] + = 1$;
- The frequency estimation for item $e_j$ is $\min_{i \in [d]} A[i][h_i(e_j)]$.

**Figure:** An illustration of the Count-Min sketch.
How to set the number $d$ of hash functions and the size $w$ of each array?

- Recap: We have a stream that includes a sequence of $n$ integers, which are from the domain $[m]$.
- We have the following lemma for the counter maintained at each row (for each hash function). Now, let’s focus on the $i$-th row (other rows can be analyzed in the same approach).
- With Lemma 5, we know how to set $w$.

**Lemma 5**

Given an arbitrary element $e$, let $A[i][h_i(e)]$ be the counter value at the $h_i(e)$-th position of the $i$-th array. Setting $w = \lceil \frac{3}{\varepsilon} \rceil$, we have that:

$$\Pr[A[i][h_i(e)] - f(e) \geq \varepsilon \cdot n] \leq \frac{1}{3}.$$
Proof.

Let $e_x$ be an arbitrary element that is not equal to $e$. Then, we know that $\mathbb{P}[h_i(e) = h_i(e_x)] = \frac{1}{w}$ since $h_i$ is a 2-wise independent hash function. Hence, let $f_{e}$ be the number of elements in the stream that is not equal to $e$. Clearly, we know $f_{e} + f(e) = n$. Let $X$ be the overestimation of $f(e)$, i.e., $A[i][h_i(e)] - f(e)$. Then, by expectation, we know

$$\mathbb{E}[X] = \frac{f_{e}}{w} \leq \frac{n}{w}.$$ 

Then, by the Markov’s inequality, we have:

$$\mathbb{P}[A[i][h_i(e)] - f(e) \geq \varepsilon \cdot n] = \mathbb{P}[X \geq \varepsilon \cdot n] \leq \frac{\mathbb{E}[X]}{\varepsilon \cdot n} \leq \frac{n}{\left\lceil 3/\varepsilon \right\rceil \cdot \varepsilon \cdot n} \leq \frac{1}{3}$$

Proof done.
How to set the number $d$ of hash functions and the size $w$ of each array?

- By Lemma 5, we know how to set $w$.
- We set $d$ to achieve $(\varepsilon \cdot n, \delta)$-approximation via Lemma 6.

**Lemma 6**

*Given an arbitrary element $e$, let $\hat{f}(e) = \min_{i \in [d]} A[i][h_i(e)]$ with the $d$ setting as $\lceil \log_3 (1/\delta) \rceil$, we have that:

$$P[\hat{f}(e) - f(e) \geq \varepsilon \cdot n] \leq \delta$$

The above is easy to prove since every row fails with at most $\frac{1}{3}$ probability and thus all $d$ rows fail with at most $\frac{1}{3^d} \leq \delta$ probability.

Total space cost: $O\left(\frac{\log m \cdot \log(1/\delta)}{\varepsilon}\right)$ bits.
Count-Min Sketch Extension: Range-based Queries

**Definition 2 (Range-based Frequency Queries)**

Given a range \([\ell \cdots r]\), the range-based frequency query asks for the frequency of the elements in \([\ell \cdots r]\), i.e., \(f(\ell \cdots r) = \sum_{i=\ell}^{r} f(i)\).

A naive solution: Let \(\hat{f}(e)\) be the estimation of the frequency \(f(e)\) of element \(e\) returned by the Count-Min sketch. Return \(\sum_{i=\ell}^{r} \hat{f}(i)\).

- **Issue**: The larger the range it is, the larger the approximation error will be. A very loose bound: \((r - \ell) \cdot \varepsilon \cdot n\).

An improved solution: Dyadic tree-based solution.

- **Assumption**: \(m\) is a power of 2. If not, we enlarge the domain from \([m]\) to \([2^{\lceil \log_2 m \rceil}]\). Let \(L = \lceil \log_2 m \rceil\).

- **Main idea**: For ranges \([1 \cdots 2], [3 \cdots 4], \cdots [2^L - 1 \cdots 2^L]\) (each pair), \([1 \cdots 4], [5 \cdots 8], \cdots [2^L - 3, \cdots 2^L]\) (each quadruple), \(\cdots, [1 \cdots 2^{L-1}], [2^{L-1} + 1 \cdots 2^L]\), we track the frequency for each range.
An improved solution: Dyadic tree-based solution (cont.).

- Build a Count-Min sketch for $[1 \cdots 1], [2 \cdots 2], \cdots, [2^L \cdots 2^L]$
- Build a Count-Min sketch for $[1 \cdots 2], [3 \cdots 4], \cdots [2^L - 1 \cdots 2^L]$.
- ...
- Build a Count-Min sketch for $[1 \cdots 2^{L-1}], [2^{L-1}, 2^L]$
- We have $L$ Count-Min sketches.
- Given an element $e$, it is easy to obtain its corresponding ranges via bit-operations. Then, we update the corresponding sketch entries.

How to Query:

- Given a range $[\ell \cdots r]$, it can be transformed into at most $2L$ disjoint ranges so that their union is $[\ell \cdots r]$. Thus, we can apply the Count-Min sketch for each range and then sum them together. The error is bounded by $2L \cdot \varepsilon \cdot n$, which is $O(\log m \cdot \varepsilon \cdot n)$. 
Definition 3 (Turnstile Model)

The stream consists of a sequence of $n$ pairs $(e_i, c_i)$, where $e_i \in [m]$ is an item and $c_i$ is the number of items to be added (when $c_i$ is positive) or deleted (when $c_i$ is negative). The count of an element cannot be negative at any stage. The frequency of an item $e$ is

$$f(e) = \sum_{(e_i, c_i) \in S \land e_i = e} c_i.$$ 

Think: What is the new approximation guarantee?

- Define $f$ as the vector of the frequency of each distinct element. Under the same choice of $d$ and $w$, the error is related to $\|f\|_1$, i.e. the sum of the frequency of each distinct element. More specifically,

$$\mathbb{P}[\hat{f}(e) - f(e) \geq \varepsilon \|f\|_1] \leq \delta$$

Sibo WANG (CUHK)
SEEM5020 Algorithms for Big Data
Fall 2023
The approximation error of Count-Sketch is related to the $L_2$-norm of $f$.

- Maintain $d$ arrays, each with a size of $w$. For each array $A[i][1\cdots w]$, two pairwise independent hash functions $h_i : [m] \rightarrow [w]$ and $g_i : [m] \rightarrow \{1, -1\}$ are associated with this array $A[i][1\cdots w]$.
- When a pair $\langle e, c_e \rangle$ comes, update the sketch as follows:

$$A[i][h_i(e)] \leftarrow A[i][h_i(e)] + g_i(e) \cdot c_e \quad \forall i \in [1\cdots d]$$

- The estimated frequency of item $e$ is the median over the $d$ arrays of $g_i(e) \cdot A[i][h_i(e)]$.

**Analysis:** left as self-exercise.
Table of Contents

1. Finding Frequent Items in Data Streams
   - Counter-based Algorithms
   - Sketch-based Algorithms

2. Querying Over a Slide Window in Streams
   - The DGIM Algorithm
Definition 4 (Sliding Window)

Given a sequence of elements in the stream, where each element $e_i$ is associated with a timestamp $t_i$ so that $t_i < t_j$ if $i < j$, a sliding window of size $N$ consists of the set of elements $\{e_{c-N+1}, e_{c-N+2}, \cdots e_c\}$ given the current timestamp $t_c$.

Interesting case:

- $N$ is so large that the data cannot be fitted into the main memory.
- Too many streams: Windows for all cannot be stored in the memory.

Application:

- The number of queries about ChatGPT in the last 1 billion searches on Google.
- The number of times product $X$ was sold in the last 1 million sales on Alibaba.
Problem 1

Given a stream of 0s and 1s, answer the query about how many 1s are there in the last \( k \) elements, where \( k \leq N \) and \( N \) is the window size.

A naive solution: Store the most recent \( N \) elements.

- When the new element comes, discard the oldest element.
- The space is too high as we have discussed.
  - We cannot get the exact answer without storing the whole window.
- We are happy with an approximate solution in most real scenarios.
The DGIM Algorithm [4]

The DGIM algorithm refers to the inventors: Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani.

- It requires only $O(\log^2 N)$ bits per stream to provide 0.5-approximation (which we will discuss next).
- The timestamp can be maintained with $\log N$ bits by taking $t_c \% N$.
- An $\varepsilon$-approximation can be further achieved by using $O\left(\frac{\log^2 N}{\varepsilon}\right)$ bits.
A key concept in DGIM: **Bucket**

- A bucket is a segment of the window ending with a 1; it is represented by a record consisting of
  - The *timestamp* of the end of the segment \(O(\log N)\) bits).
  - The *size* of the bucket: The number of 1s in this segment.

**Constraints on the buckets:**
- The size of buckets: It must be a power of 2.
- Either one or two buckets have the same size.
- Buckets do not overlap in timestamps.
- Buckets are sorted in size where earlier buckets never have a size smaller than that of later buckets.
- A bucket disappears when the latest timestamp is not within the time window.
- We have at most \(2\log_2 N\) buckets.
An Example of Bucketized Summary of the Stream

Different buckets of the form: \(<timestamp\ of \ last\ 1, size>\), assuming that the oldest timestamp is 1 and \(N = 70\) in the above example:

- **Orange:** One orange color bucket of size 16, \(<16,16>\). Part of the 1s in the bucket is outside the window and is not shown.
- **Pink:** Two pink color buckets of size 8, \(<31,8>\) and \(<45,8>\).
- **Magenta:** Two magenta color buckets of size 4, \(<53,4>\) and \(<59,4>\).
- **Cyan:** One cyan color bucket of size 2, \(<65,2>\).
- **Yellow:** Two yellow color buckets of size 1, \(<66,1>\) and \(<69,1>\).

The above example satisfies the constraints of buckets mentioned in the previous page.
Processing the stream with the DGIM algorithm.

- **Step 1:** When a new element comes at the current timestamp, **Drop the last (oldest) bucket** if its timestamp of the ending segment is prior to \( N \) time units before the current time.
- **Step 2:** If the current element is 0. No other change is required. Otherwise, if the current element is 1:
  - **Step 2.1:** Create a new bucket of size 1 (including only this element), and the timestamp is the current timestamp.
  - **Step 2.2:** If there are no more than 2 buckets of size 1, nothing needs to be done. Otherwise, there are 3 buckets of size 1, a violation of the constraint; merge the oldest two buckets of size 1 into a size 2 bucket. We repeat the above merging steps until there is no violation of the bucket constraint (there are no more than 2 buckets of the same size).
Example: Updating the Buckets

Initial state of the stream: size-16 bucket (orange) shows partial elements in the bucket

```
1001010110001011 0 101010101010111 0 1010101
110101 000 101 1 00 1 0
```

1 arrives; create a third bucket of size 1.

```
1001010110001011 0 101010101010111 0 1010101
110101 000 101 1 00 1 0 1
```

Combine the oldest two size-1 buckets into a size-2 bucket.

```
1001010110001011 0 101010101010111 0 1010101
110101 000 101 1001 0 1
```

Later, 1, 0, 1 arrive. Now we have 3 size-1 buckets again. Need to merge until no violation on the buckets.

```
1001010110001011 0 101010101010111 0 1010101
110101 000 101 1001 0 1 1 0 1
```
Answering the Query

To estimate the number of 1s in the most recent $N$ elements

- Sum the sizes of all buckets except the oldest bucket.
- Add half the size of the oldest bucket.

Why only half of the oldest bucket?

- We do not know how many 1s of the oldest bucket are within the time window.

What is the estimated number of 1s in the time window?

- $8 + 8 + 4 + 4 + 2 + 1 + 1 + \frac{16}{2} = 36$. 
Why it can achieve a 0.5-approximation?

- Suppose the oldest bucket has a size of $2^r$.
- By assuming that half of the elements, i.e., $2^{r-1}$ elements, in the last bucket, are in the window, the error caused by the last bucket is at most $2^{r-1}$.
- Since there is at least one bucket of each size less than $2^r$ and the oldest bucket includes at least 1 element that is within the window size, the true size is at least:

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

- The error is thus bounded by $2^{r-1}/2^r = 0.5$. 
Extensions

- How to deal with queries with an arbitrary $k \leq N$?
- How to reduce the error?
- How to handle streams of arbitrary non-negative integers?

