# Computational Frameworks: Streaming

(Part 1)

# OUTLINE

1. Introduction to the streaming model
2. Warm-up: finding the majority element
3. Sampling
   - Uniform sampling with reservoir sampling
   - Finding (approximate) frequent items
4. Sketching (Part 2)
   - Counting distinct elements with probabilistic counting
   - Estimating individual frequencies and the second moment
5. Filtering: Bloom filters for membership problem (Part 2)

# Dealing with volume and velocity

**Typical scenario:**

- The data to be processed arrive as a continuous stream
  - because they are generated by some evolving (possibly endless) process;
  - or because their massive volume discourages random accesses.
- Therefore: data analysis must happen on the fly using limited memory, without storing all data for subsequent offline processing.
- Many applications (some examples next)

# Some applications

**Network management:**

- Stream: packets routed through a router.
- Task: gather traffic statistics (e.g., average number of connections/second to same IP address)

**Online auctions:**

- Stream: online auctions generate data on objects on sale, bids, . . .
- Tasks: statistical estimation of auction data (e.g., average selling price over the items sold by each seller, highest bid in a given period of time,. . . )

# Some applications

**Internet of Things:**

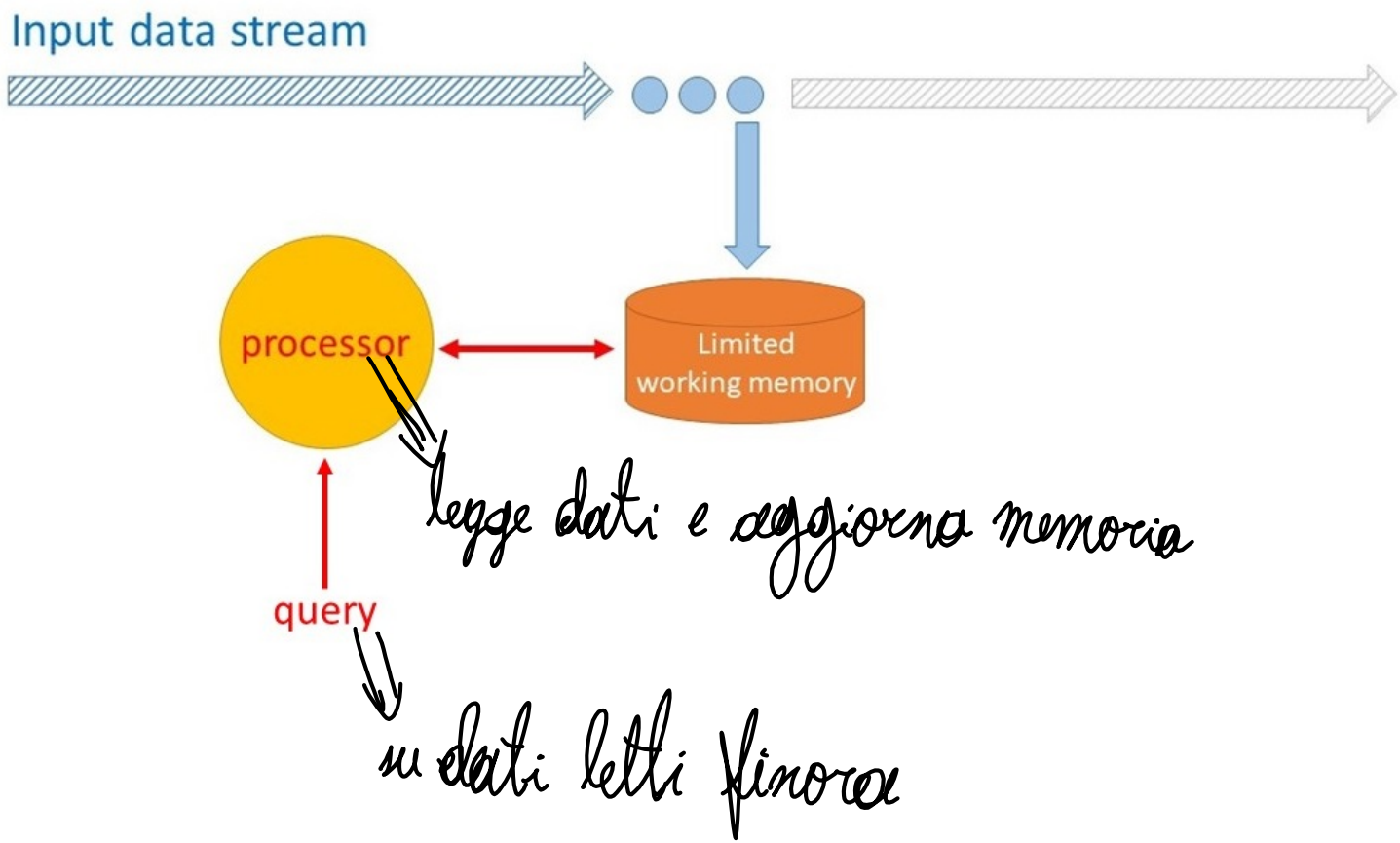
- Stream: data generated by thousands/millions of sensors.
- Tasks: learn from data, outlier detection, gather statistics,...

**Sequential disk accesses:**

- Stream: data from a massive file on secondary storage (sequential access rates are higher than random access),
- Tasks: data processing of data in the file. In this case it is also possible to sequentially scan the same file twice or even more times (multi-pass).

# Streaming Model

- (Sequential) machine with limited amount of working memory.
- Input provided as a continuous (one-way) stream.



# Streaming Model

Let $\Sigma = x_1, x_2, \ldots x_n \ldots$ denote the input stream received sequentially, where $x_i$ is the $i$th element received. Upon receiving $x_i$:

- Suitable data structures stored in the working memory are updaded (**UPDATE task**)
- If required, a solution for the problem at hand, relative to the input set $x_1, X_2, \ldots, x_i$ is computed from the data stored in the working memory (**QUERY task**)

The following **performance metrics** are usually considered

- Metric 1: Size $s$ of the working memory (aim $s \ll |\Sigma|$)
- Metric 2: Number $p$ of sequential passes over $\Sigma$ (aim: $p = 1$)
- Metric 3: Processing time per item $T$ (aim: $T = O(1)$)

# Streaming Model

## Algorithm Design Techniques

- Approximate solutions (exact ones may require linear space)
- Mantain lossy summary of $\Sigma$ via a synopsis data structure (e.g., random sample, hash-based sketch)

non generali, ma specifiche per problema

**Typical data analysis tasks**

- Identification of frequent items.
- Useful statistics: e.g., frequency moments, quantiles, histograms
- Optimization and graph problems: e.g., clustering, triangle counting

**Goal:** suitable tradeoffs between accuracy, working memory, and processing time per item (i.e., *throughput*).

# Warm-up:

# finding the majority element in a stream

# Warm-up: finding the majority element

**Majority problem**

Given a stream $\Sigma = x_1, x_2, \ldots, x_n$, return the element $x$ (if any) that occurs $> n/2$ times in $\Sigma$.

**Examples:**

- In $\Sigma = A, B, A, C, A, D, A, A$, the majority element is $A$.
- In $\Sigma = A, B, C, D, D, E, E$, a majority element does not exist.

**Standard off-line setting.** Easily solved using linear space in $O(n \log n)$ time, through sorting, or $O(n)$ expected time, through hashing.

**Streaming setting.** Need 1 or 2 passes, depending on goals

- First pass (Boyer-Moore algorithm): find an element $x$ which is the majority element, if one exists.
- Second pass: check whether $x$ is indeed the majority element.

# Boyer-Moore algorithm

perso, alg. Streaming hanno poche righe

The Boyer-Moore algorithm maintains 2 integers *cand* and *count*:

- *cand* contains a candidate majority element (*the true majority element, if one exists*);
- *count* is a counter;

**Initialization:** $cand \leftarrow$ null and $count \leftarrow 0$.

**For each** $x_t$ in $\Sigma$ **do**
  **if** $count = 0$ **then** $\{cand \leftarrow x_t;\ count \leftarrow 1;\}$
  **else** {
    **if** $cand = x_t$ **then** $count \leftarrow count + 1$
    **else** $count \leftarrow count - 1$
  }

UPDATE

**At the end: return** *cand*

QUERY

# Example

$\Sigma = A, A, A, C, C, B, B, A, A$

| | CAND | COUNT |
|---|---|---|
| | null | 0 |
| A | A | 1 |
| A | A | 2 |
| A | A | 3 |
| C | A | 2 |
| C | A | 1 |
| B | A | 0 |
| B | B | 1 |
| A | B | 0 |
| A | A | 1 ⇒ valore finale non garantisce correttezza |

candidato: A

verificare: guardo tutto stream

impossibile per risultati real-time

# Example

$\Sigma = A, A, A, C, C, C, B, B, B$

| | CAND | COUNT |
|---|---|---|
| | null | 0 |
| A | A | 1 |
| A | A | 2 |
| A | A | 3 |
| C | A | 2 |
| C | A | 1 |
| C | A | 0 |
| B | B | 1 |
| B | B | 2 |
| B | B | 3 |

# Warm-up: finding the majority element

**Theorem**

*Given a stream $\Sigma$ which contains a majority element $m$, the Boyer-Moore algorithm returns $m$ using:*

- *working memory of size $O(1)$;* ⇒ solo 2 interi
- *1 pass;* ⇒ solo primo step; assumiamo esista elemento
- *$O(1)$ time per element.* ⇒ solo update

Più difficile dimostrare correttezza

Dimostrare $CAND_n = m$ alla fine

$\Downarrow$ $\Rightarrow$

# el. in stream vero majority el.

Assumiamo che $CAND_n \neq m$ alla fine
$CAND_i :=$ candidato dopo iterazione $i$ (0: inizio)

Consideriamo ultima it. $CAND_n = p \neq m$, $COUNT_n = k$
Partiziono $\Sigma$:
- $COUNT_n = k$ elementi uguali a $CAND_n = p$
- altri $n-k$ elementi $\Rightarrow$ raggruppabili a coppie $(x_i, x_j)$, $x_i \neq x_j$ $\Rightarrow$
  $\Rightarrow$ un elemento aumenta COUNT, altro lo riduce

Assumiamo $p \neq m \Rightarrow m$ può apparire $l = \frac{n-k}{2} \in \left[0, \frac{n}{2}\right]$ volte ($\Leftrightarrow k \in [0, n]$)

$\Downarrow$

contraddizione:
$m$ majority el $\Rightarrow l > \frac{n}{2}$ ∎

## Exercise

Let $\Sigma = x_1, x_2, \ldots x_n$ be a stream of $n$ distinct integers in $[1, n+1]$. Design a streaming algorithm that finds the missing number using $O(1)$-size working memory, 1 pass, and $O(1)$ time per element.

# Sampling

# Sampling

buona rappresentazione di $X$

**Definition**

Given a set $X$ of $n$ elements and an integer $1 \leq m < n$, an $m$-sample of $X$ is a random subset $S \subset X$ of size $m$, such that for each $x \in X$, we have $\Pr(x \in S) = m/n$ (uniform sampling).

**In the off-line setting**, an $m$-sample $S$ of $X$ can be easily computed by selecting $m$ elements from $X$ (or, equivalently, their indices) with uniform probability and without replacement.

**In the streaming setting**, things get harder, especially in application scenarios where streams are potentially unbounded and at every time step a random sample of the elements seen so far is required (e.g., *monitoring of sensor data*)

+ semplice: prima di streaming scelgo sample (indici) => richiede sapere $n$

# Sampling

## Sampling problem

Given a (possibly unbounded) stream $\Sigma = x_1, x_2, \ldots$ and an integer $m < |\Sigma|$ maintain, for every $t > m$, an $m$-sample $S$ of the prefix $\Sigma_t = x_1, x_2, \ldots, x_t$. $\Rightarrow \Pr(x_i \in S) = \frac{m}{t}$ $\forall$ istante $t \in [1, n]$

For a fixed $t$, the solution is easy

- Before the stream starts, compute an $m$-sample $\mathcal{I}$ of the set of integers $\{1, \ldots t\}$. $\Rightarrow$ indici
- For each entry $x_i \in \Sigma_t$ with $i \in \mathcal{I}$, add $x_i$ to $S$.

**How can we maintain $S$ for every $t$?**

# Reservoir Sampling: algorithm

The following algorithm, dubbed Reservoir Sampling was developed by J.S. Vitter in 1985.

⇒ introdotto per modelli di memoria esterna (e.g. nastri)

**Initialization:** $S \leftarrow \emptyset$.

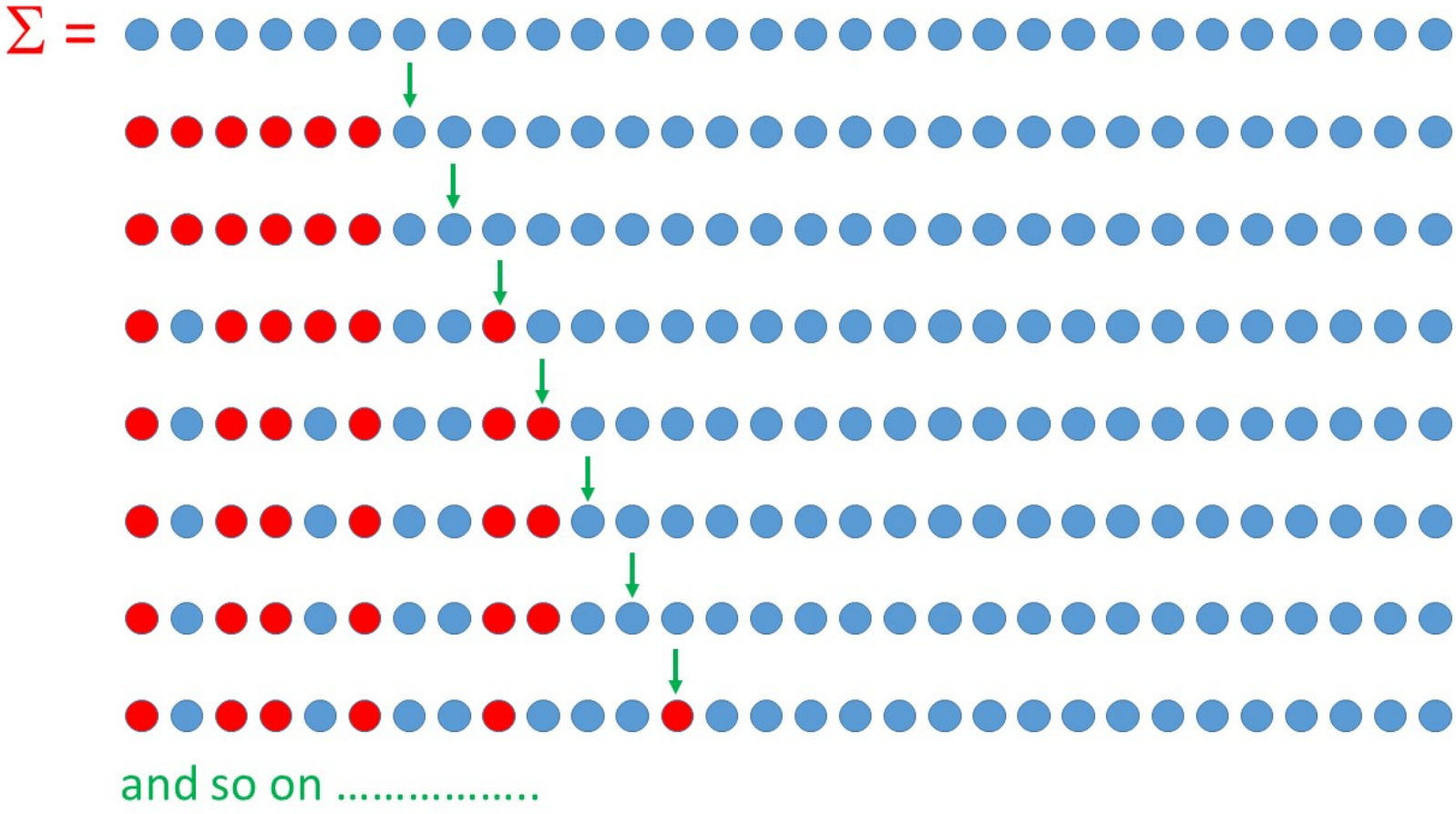
**For each** $x_t$ in $\Sigma$ **do**
  **if** $t \leq m$ **then** add $x_t$ to $S$;
  **else** with probability $m/t$ do the following:
    evict a random element $x$ from $S$
    add $x_t$ to $S$
  }

⇒ all'inizio prendiamo tutto
⇒ t cresce, prob. scende
⇒ sostituiamo elementi

m-esimo elemento sarà in sample $(m/m)$ ⇒ proprietà soddisfatta

Remark: the stream size needs not be known, and, in fact, it can be unbounded.

# Example: $m = 6$



# Reservoir Sampling: analysis

**Theorem**

*Let $\Sigma = x_1, x_2 \ldots$. For any time $t \geq m$, the set $S$ maintained by the reservoir sampling algorithm is an $m$-sample of $\Sigma_t = x_1, x_2, \ldots, x_t$.* $\Rightarrow \forall t \geq m,\ \Pr[x_i \in S] = m/t \ \forall i \in [1, t]$

1 pass, memoria dim. $m$, processing time costante (dipende da implementazione $S$)

Induzione:

- base: $t = m \Rightarrow x_1, \dots, x_m$ in $S$ con prob. $1 = \frac{m}{m} \Rightarrow$
  $\Rightarrow S$ è $m$-sample per $x_1, \dots, x_m$
- induzione: $t \geq m$
  1) $\Pr[x_t \in S] = m/t$ (ultimo el.) $\Rightarrow$ per costruzione
  2) $\Pr[x_i \in S] \; \forall i \leq t$

  Sia $S_h$ sample $S$ dopo aver ricevuto $x_1, \dots, x_h$

$$\Pr[x_i \in S] = \Pr[x_i \in S_t \mid x_t \in S_t] \Pr[x_t \in S_t] + \Pr[x_i \in S_t \mid x_t \notin S_t] \Pr[x_t \notin S_t]$$

$$\Pr[x_i \in S_t \mid x_t \in S_t] = \Pr[x_i \in S_{t-1}] \Pr[x_i \text{ non cacciato a tempo } t \mid x_i \in S_{t-1}]$$

$$= \frac{m}{t-1}\left(\frac{m-1}{m}\right) = \frac{m-1}{t-1}$$

$$\Pr[x_i \in S_t \mid x_t \notin S_t] = \Pr[x_i \in S_{t-1}] = \frac{m}{t-1}$$

$$P_2[x_i \in S] = \frac{n-1}{t-1} \cdot \frac{n}{t} + \frac{n}{t-1} \cdot \left(1 - \frac{n}{t}\right) = \frac{n(n-1) + n(t-n)}{t(t-1)} =$$

$$= \frac{n^2 - n + tn - n^2}{t(t-1)} = \frac{n}{t}$$

# Sampling-based applications: Frequent Items

**Frequent Items problem**

Given a stream $\Sigma = x_1, x_2, \ldots, x_n$ of $n$ items and a frequency threshold $\varphi \in (0, 1)$, determine all distinct items that occur at least $\varphi \cdot n$ times in $\Sigma$ (we call them frequent items).

$x_i \in U$ universo

$\varphi = 1/2 \Rightarrow$ max. 1 el.

Usare reservoir sampling

1 - calcolare $m$-sample
2 - ritornare el. unici in $m$-sample come frequent items

$m$ piccolo ($m << \frac{1}{\varphi}$) $\Rightarrow$ no spazio per tutti f.i.
$m$ grande ($m >> 1/\varphi$) $\Rightarrow$ troppo rumore

$m = 1/\varphi$
$\forall$ f.i. $f \in U$, $\mathbb{E}[\vec{m}_f] = 1$ ($\vec{m}_f$ = #occorrenze $f$ in sample)
($n_f$ = # $f$ in $\Sigma \geq \varphi n$)

$x_{i_1}, \ldots, x_{i_{n_f}}$ : el. in $\Sigma$ uguali a $f$
$\rightarrow$ tutti con prob. $m/n$ di essere in sample

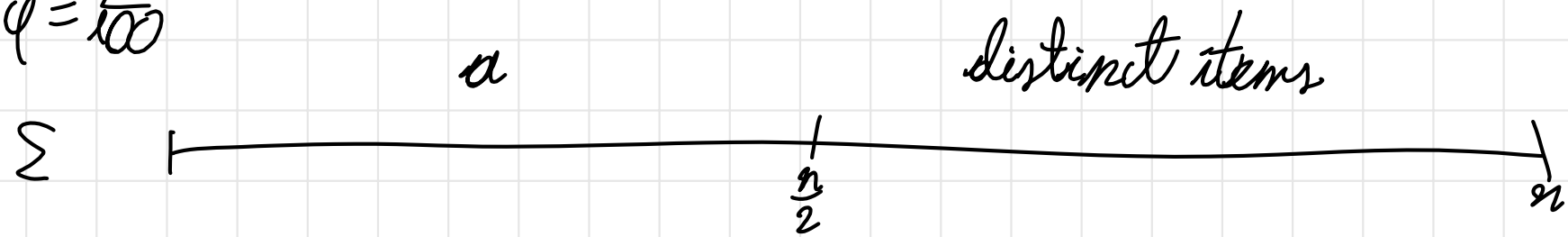
$m_f = \sum_{j=1}^{n_f} y_j \Rightarrow y_j = \begin{cases} 1 & \text{se } x_{i_j} \text{ in sample} \\ 0 & \text{altrimenti} \end{cases}$

$$\mathbb{E}[m_k] = \mathbb{E}\left[\sum_{j=1}^{n_k} Y_j\right] = \sum_{j=1}^{n_k} \mathbb{E}[Y_j]$$

$$\mathbb{E}[Y_j] = 1 \cdot \Pr[x_{ij} \text{ in sample}] + 0 \cdot \Pr[x_{ij} \text{ not in sample}] = \frac{m}{n}$$

$$\mathbb{E}[m_k] = n_k \cdot \frac{m}{n} \geq \varphi n \cdot \frac{1}{\varphi n} \geq 1$$

$\varphi = \frac{1}{100}$




m - sample $\Rightarrow m = 100$

↳ meta a, meta distinct items

Reservoir sampling può trovare molto rumore

# Frequent Items with Reservoir sampling

Frequent items can be approximated through reservoir sampling:

1. Extract an $m$-sample $S$ from $\Sigma$ with reservoir sampling. Note that the same element may occurr multiple times in the sample!
2. Return the subset $S' \subseteq S$ of distinct items in the sample

**How well does $S'$ approximate the set of frequent items?**

- If $m \geq 1/\varphi$, for any given frequent item $a$, we expect to find at least one copy of $a$ in $S$ (hence in $S'$).
- However, some frequent item may be missed, and $S'$ may contain items with very low frequency.

# Approximate Frequent Items

**$\epsilon$-Approximate Frequent Items ($\epsilon$-AFI) problem**

Given the stream $\Sigma = x_1, x_2, \ldots, x_n$ of $n$ items, a frequency threshold $\varphi \in (0, 1)$, and an accuracy parameter $\epsilon \in (0, \varphi)$, return a set of distinct items that

- includes all items occurring at least $\varphi \cdot n$ times in $\Sigma$.
- contains no item occurring less than $(\varphi - \epsilon) \cdot n$ times in $\Sigma$.

no falsi negativi

no elementi rari

**Remark:** with this formulation, we avoid false negatives but tolerate high-frequency false positives.

Risolviamo $\varepsilon$-AFI con prob. $\geq 1-\delta$

$\Downarrow$

con prob. $< \delta$ non riusciamo a trovare sol.

Reservoir sampling
- tutti frequent items se $m \sim \frac{1}{\varepsilon} \log n$
- no rare items non vale (possiamo garantirlo solo con 2 pass)

# Sticky Sampling

Sticky Sampling provides a probabilistic solution to the $\epsilon$-AFI problem.

MAIN INGREDIENTS:

per garanzie teoriche

- Confidence parameter $\delta \in (0, 1)$.
- Hash Table $S$, whose entries are pairs $(x, f_e(x))$ where
  - $x$ (the key) is an item
  - $f_e(x)$ (the value) is an lower bound to the number of occurrences of $x$ seen so far.
- Sampling rate chosen to ensure (with probability $\geq 1 - \delta$) the frequent items are in the sample.

# Sticky Sampling: algorithm

Consider stream $\Sigma = x_1, x_2, \ldots, x_n$, and assume $n$ known!

**Initialization:**

- $S \leftarrow$ empty Hash Table);
- $r = \ln(1/(\delta\varphi))/\epsilon$; // *sampling rate* $= r/n$.

**For each** $x_t$ in $\Sigma$ **do**

**if** $(x_t, f_e(x_t)) \in S$ **then** $f_e(x_t) \leftarrow f_e(x_t) + 1$;
**else** add $(x_t, 1)$ to $S$ with probability $r/n$; /* (*start tracking* $x_t$) */

**At the end: return** all items $x$ in $S$ with $f_e(x) \geq (\varphi - \epsilon)n$

# Sticky Sampling. analysis

**Theorem**

*Sticky sampling solves the $\epsilon$-AFI problem correctly with probability at least $1 - \delta$ and requires*

- *working memory of size $O\left(\ln(1/(\delta\varphi))/\epsilon\right)$, in expectation;*
- *1 pass;*
- *$O(1)$ expected processing time per element.*

Remark: The space is independent of the stream length.

Dimostriamo correttezza e working memory (altri errori)

CORRETTEZZA per $\epsilon$-AFI:
1 - no falsi negativi
2 - no rare items

2- Abbiamo $f(x) \geq f_e(x)$
Supponiamo $f_e(x) = 0$ se $x \notin S$
Se $x = f(x) < (\varphi - \epsilon) n$, allora $f_e(x) \leq f(x) < (\varphi - \epsilon) n \Rightarrow$ non ritornato

1 - sia $x$ frequent item ($f(x) \geq \varphi n$) $\Rightarrow$ quando non ritornato? $\Rightarrow$
$\Rightarrow$ quando non lo prendiamo noi o non raggiungiamo soglia
$\Rightarrow$ per contiamo le prime $f(x) - (\varphi - \epsilon) n$ volte $\Rightarrow$
$\Rightarrow f_e(x) < (\varphi - \epsilon) n \Rightarrow$ non ritorniamo $\Rightarrow$

$\geq \varphi n - (\varphi - \epsilon) n > \epsilon n$

$\mathbb{P}[\text{no sampling prime } \in n \text{ occ.}] =: p$

$p = \left(1 - \frac{r}{n}\right)^{\varepsilon n} = \left(1 - \frac{1}{\frac{n}{r}}\right)^{\frac{n}{r}\varepsilon n} \leq e^{-\varepsilon n \frac{r}{n}} \leq e^{-\varepsilon r}$

$\forall x \geq 1 \quad \left(1 - \frac{1}{x}\right)^x \leq e^{-1}$

$\left(x = \frac{n}{r}\right)$

$\Pr[\text{no ritorno } x \text{ come frequent item}] \leq p \leq e^{-\varepsilon r}$

$\Pr[\text{sticky sampling fallisce}] = \Pr[\text{almeno un frequent item non ritornato}]$

$\leq \sum_{x \in F} \Pr[\text{non ritorno } x \text{ come f. i.}] \leq \frac{1}{\varphi} e^{-\varepsilon r} \leq \delta$

$e^{-\varepsilon r} \leq \delta\varphi \Rightarrow \frac{1}{\varphi\delta} \leq e^{\varepsilon r} \Rightarrow \log \frac{1}{\varphi\delta} \leq \varepsilon r \Rightarrow r \geq \frac{1}{\varepsilon} \log \frac{1}{\varphi\delta}$ ■

# WORKING MEMORY (EXPECTED) (upper bound): $O(r)$

Caso peggiore: tutti d. diversi

working memory $= O(|S|) = \sum_{x \in \Sigma} Y_x$

$=$

$\begin{cases} 1 & \text{if } x \in S \\ 0 & \text{else} \end{cases}$

$\mathbb{E}[\text{working memory}] = \mathbb{E}\left[\sum_{x \in \Sigma} Y_x\right] = \sum_{x \in \Sigma} \mathbb{E}[Y_x] = n \cdot \frac{r}{n} = r \Rightarrow O(r)$ ■

# Sticky Sampling with unknown $n$

Suppose that $n$ is not known by the algorithm or $\Sigma$ is unbounded.

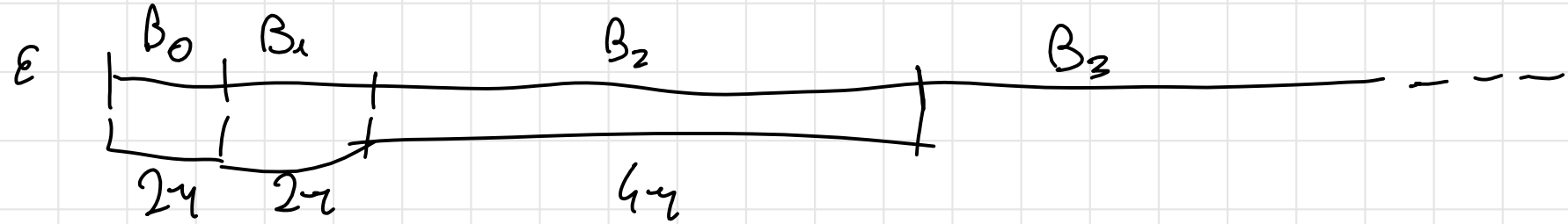
**GOAL:** After processing $x_1, x_2, \ldots, x_n$ we want to be able to derive a solution to the $\epsilon$-AFI problem for $x_1, x_2, \ldots, x_n$, for every $n \geq 1$.

IDEA. Maintain sampling rate always $\geq r/n$, with

$$r = \ln(1/(\delta\varphi))/\epsilon,$$

as follows.

- Subdivide the stream $\Sigma$ into batches $B_0, B_1, B_2, \ldots$ of geometrically growing size: namely: $|B_0| = 2r$ and $|B_i| = 2^i r$, for every $i \geq 1$.
- Sample each batch $B_i$ with geometrically decreasing rate $1/2^i$.
- Recalibrate $S$ at the beginning of each batch.




S.R. $\quad 1 \quad \frac{1}{2} \quad \frac{1}{4}$

$S_0 \Rightarrow S_1 \longrightarrow S_2$

$\Downarrow$ tengo segni ed.

$\downarrow$ tengo con prob. 1/2

# Sticky Sampling with unknown $n$: algorithm

**Initialization:**

- $S \leftarrow$ empty Hash Table);
- $r = \ln(1/(\delta\varphi))/\epsilon$;

**For each** $x_t \in \Sigma$ **do**

Let $B_i$ the batch of $x_t$;
**if** $x_t$ is the first element of $B_i$ **then** {
  **For each** $(x, f_e(x)) \in S$ **do**
    Let $\tau =$ number of tails before head of an unbiased coin.
    **if** $f_e(x) - \tau > 0$ **then** $f_e(x) = f_e(x) - \tau$
    **else** delete $(x, f_e(x))$ from $S$
}
**if** $(x_t, f_e(x_t)) \in S$ **then** $f_e(x_t) \leftarrow f_e(x_t) + 1$;
**else** add $(x_t, 1)$ to $S$ with probability $1/2^i$;

prob. 1/2

# Observations

$f_e(x)$

flip coin
- prob. 1/2 — $f_e(x) = f_e(x) - 1$
- prob. 1/2 — stop; tengo $x$ in $5$ se $f_e(x) > 0$

dimostrabile $\frac{1}{2^i} \sim \frac{1}{n}$

## Exercise

Let $\Sigma = x_1, x_2, \ldots x_n$ be a stream of $n$ items and let $S$ be an $m$-sample computed on $\Sigma$ using reservoir sampling. For a given frequency threshold $\phi \in (0, 1)$, consider a frequent item $a$, that is an item occurring at least $\phi n$ times in $\Sigma$. Show that, if $m \geq /\phi$, at least one occurrence of $a$ is in $S$, in expectation.

## Exercise

For a stream $\Sigma$ of $n$ items $x_1, x_2, \ldots, x_n$, consider the following algorithm:

- Run in parallel the Boyer-Moore majority algorithm and the Sticky Sampling algorithm with $\varphi = 1/2$.
- Let cand be the candidate identified by the Boyer-Moore algorithm, and $S$ the set of items returned by the Sticky Sampling algorithm.

**If** cand $\in S$ **then return** cand **otherwise return** null.

What can we say about the output of the algorithm? Distinguish among the case where $\Sigma$ contains a majority element and the case where $\Sigma$ does not contain a majority element.

# References


- **[LRU14]** J. Leskovec, A. Rajaraman and J. Ullman. Mining Massive Datasets. Cambridge University Press, 2014. Chapter 4 (Sections 4.3-4.5) (pdf provided in Moodle)

- **[DF08]** C. Demetrescu and I. Finocchi. Algorithms for Data Streams. In *Handbook of Applied Algorithms*, Chapter 8, Section 3. Wiley-IEEE Press, 2008. (pdf provided in Moodle)
- **[CGHJ12]** G. Cormode, M.N. Garofalakis, P.J. Haas, C. Jermaine. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. Foundations and Trends in Databases 4(1-3): 1-294, 2012. Chapter 5 (Sections 5.1-5.3) (pdf provided in Moodle)