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:

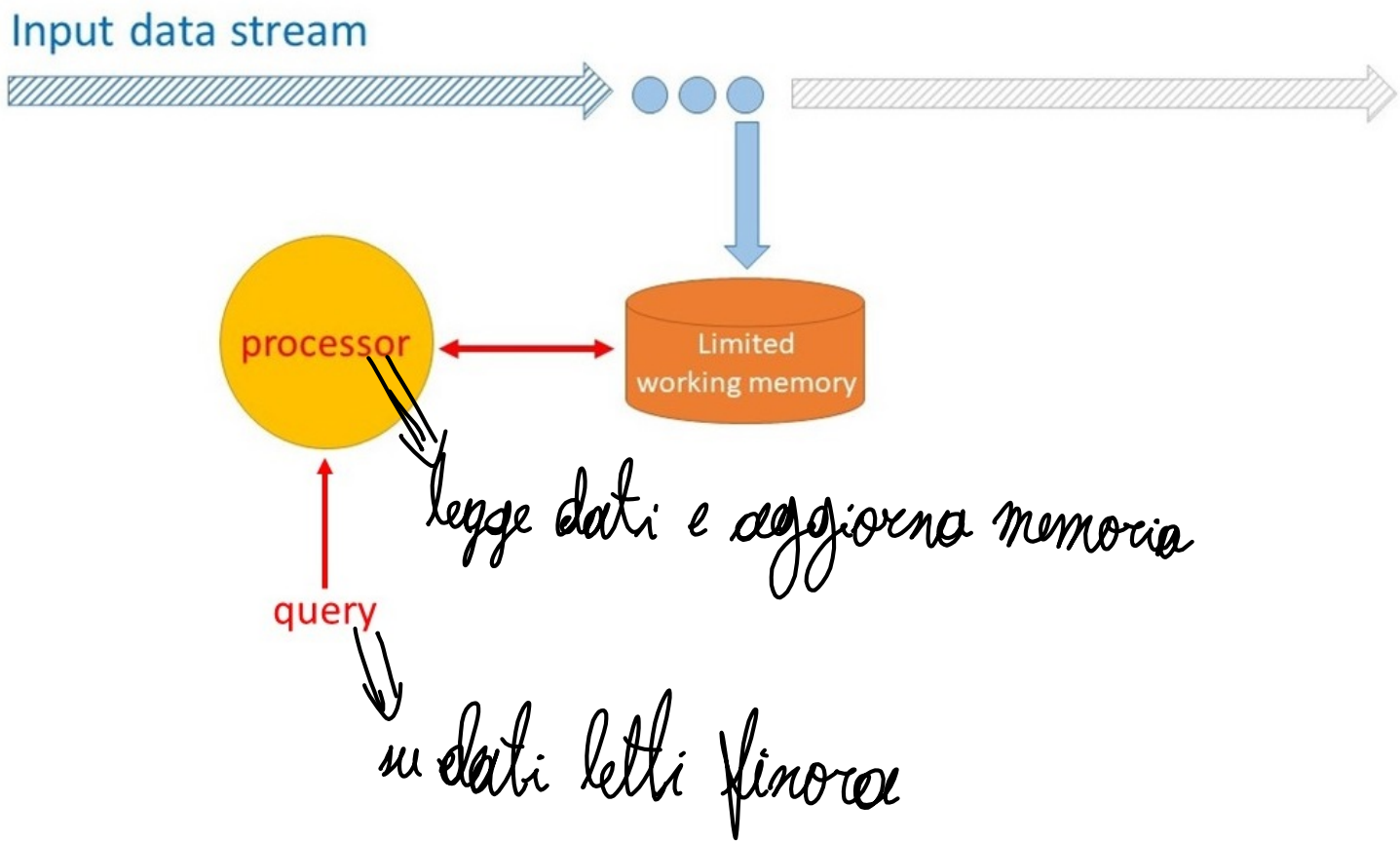
- Stream: data generated by thousands/millions of sensors.
- Tasks: learn from data, outlier detection, gather statistics,...

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 ith 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 Σ (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 Σ 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 Σ.

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 Σ **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 Σ 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 Σ:
- $COUNT_n = k$ 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

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*)

Sampling

Sampling problem

Given a (possibly unbounded) stream $\Sigma = x_1, x_2, \ldots$ and an integer $m < |\Sigma|$, maintain, for every $t \geq m$, an m-sample S of the prefix $\Sigma_t = x_1, x_2, \ldots, x_t$.

For a fixed t, the solution is easy

- Before the stream starts, compute an m-sample $\mathcal{I}$ of the set of integers $\{1, \ldots t\}$.
- 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

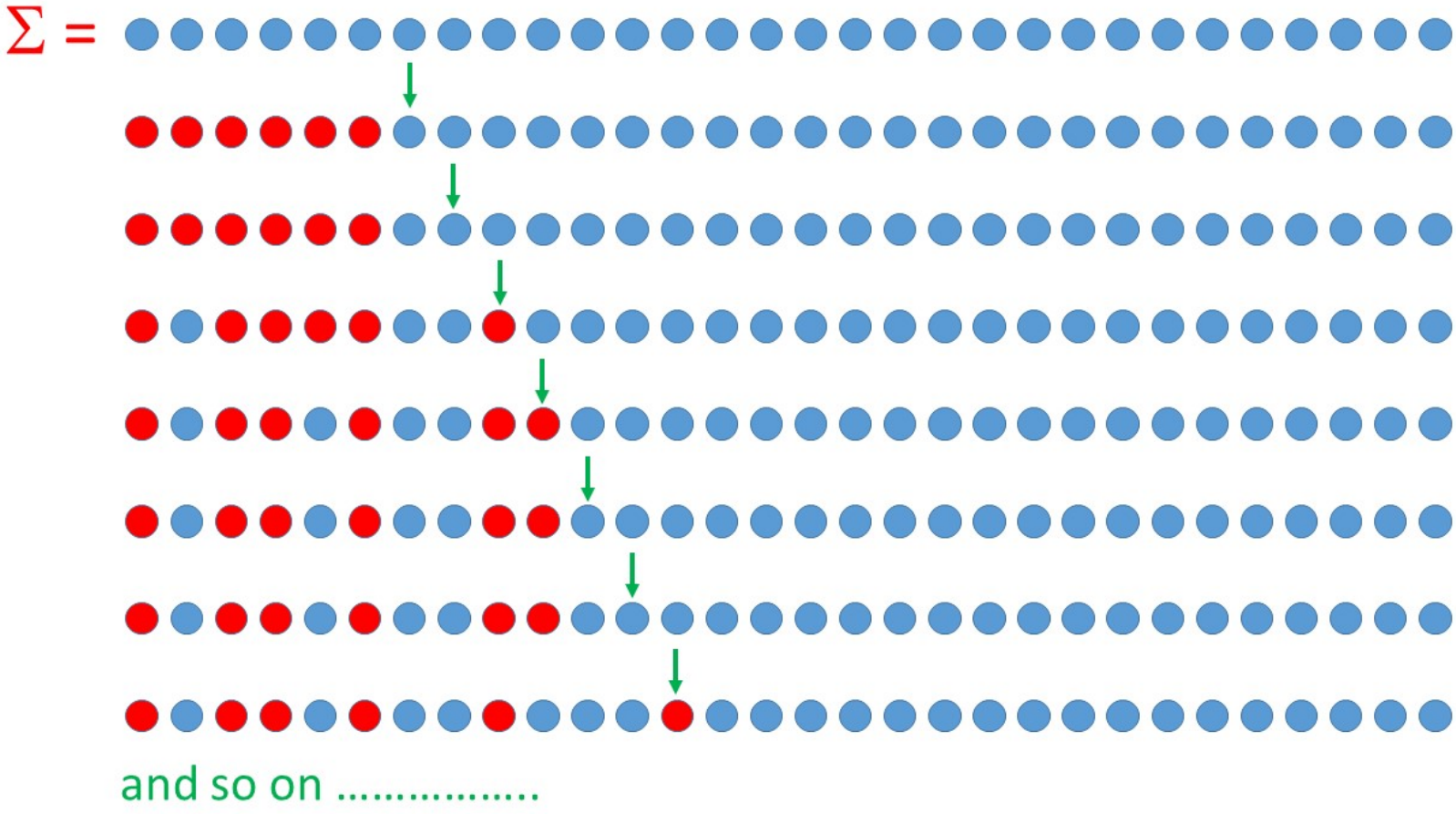
The following algorithm, dubbed Reservoir Sampling was developed by J.S. Vitter in 1985.

Initialization: $S \leftarrow \emptyset$.

For each x_t in Σ **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
  }

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 $\Sigma_t = x_1, x_2, \ldots, x_t$.

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 Σ (we call them frequent items).

Frequent Items with Reservoir sampling

Frequent items can be approximated through reservoir sampling:

1. Extract an m-sample S from Σ 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

ϵ-Approximate Frequent Items (ϵ-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 Σ.
- contains no item occurring less than $(\varphi - \epsilon) \cdot n$ times in Σ.

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

Sticky Sampling

Sticky Sampling provides a probabilistic solution to the ϵ-AFI problem.

MAIN INGREDIENTS:

- 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 Σ **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 ϵ-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.

Sticky Sampling with unknown n

Suppose that n is not known by the algorithm or Σ is unbounded.

GOAL: After processing $x_1, x_2, \ldots, x_n$ we want to be able to derive a solution to the ϵ-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 Σ 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.

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 $1/2^i$;

Observations

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 Σ using reservoir sampling. For a given frequency threshold $\phi \in (0, 1)$, consider a frequent item a, that is an item occurring at least ϕn times in Σ. Show that, if $m \geq /\phi$, at least one occurrence of a is in S, in expectation.

Exercise

For a stream Σ 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 Σ contains a majority element and the case where Σ 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)