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 frequent items with sticky sampling
4. Sketching
   - Counting distinct elements with probabilistic counting

Introduction to the streaming model

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:

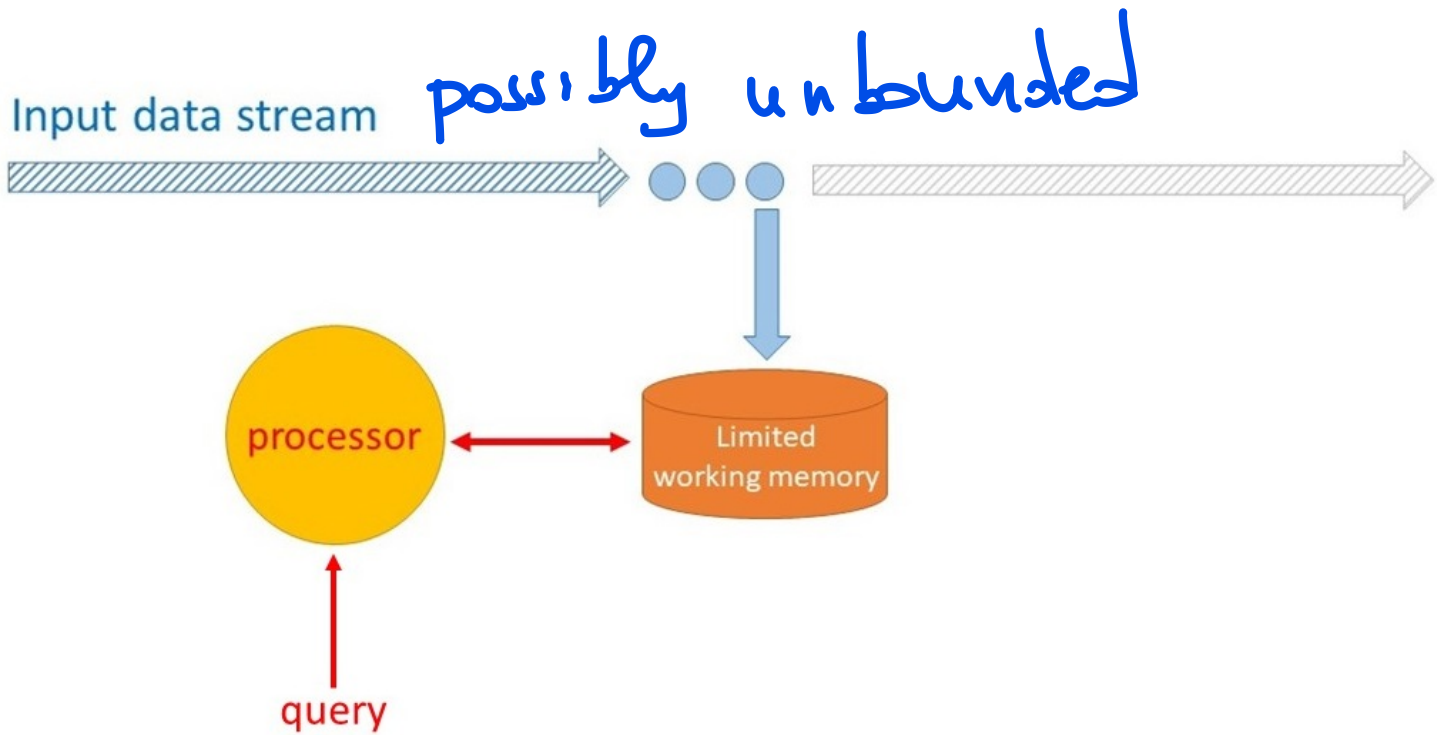
- Stream: data generated by thousands/millions of sensors.
- Tasks: learn from data, outlier detection, gather statistics,...

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

The Model

- Input stream $\Sigma = x_1, x_2, \ldots x_n$ accessed/processed sequentially
- Metric 1: Size s of the working memory (aim: $s \ll n$)
- Metric 2: Number p of sequential passes over Σ (aim: $p = 1$)
- Metric 3: Processing time per item T (aim: $T = O(1)$)

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)

Typical data analysis tasks

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

Goal: suitable tradeoffs between accuracy, required working memory, and processing time per element (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

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: $cand \leftarrow \text{null}$ and $count \leftarrow 0$.

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

Obs. count is ALWAYS ≥ 0

At the end: return *cand*

Example

$x_1\ x_2\ x_3 \qquad\qquad x_9$

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

STEP	cand	count
0	null	0
1	A	1
2	A	2
3	A	3
4	A	2
5	A	1
6	A	0
7	B	1
8	B	0
9	A	1

At the end: cand = A which is the true majority

Example

$x_1\ x_2\ x_3 \qquad\qquad x_9$

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

STEP	cand	count
0	null	0
1	A	1
2	A	2
3	A	3
4	A	2
5	A	1
6	A	0
7	B	1
8	B	2
9	B	3

At the end cand = B but then is no majority

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 $O(1)$;*
- *1 pass;*
- *$O(1)$ time per element.*

Proof

* Working memory $O(1)$
* 1 pass
* $O(1)$ time per element

} Straightforward

CORRECTNESS: If there exists a majority element m, then the algorithm returns m

Suppose that we observed $x_1 x_2 .. x_t$ for some $t \in [0, n]$ ($t=0$ is the beginning of the stream, before x_1 arrives)

what do cand and count represent at this point?

Indeed, $x_1, x_2, \ldots, x_t$ can be partitioned into

P
* count occurences of cand (perhaps more exist)
* $(t - count)/2$ pairs (e_1, e_2) with $e_1 \neq e_2$
  (some of these pairs may contain cand)

In the first example, when $t = 5$ we have

$x_1 \ldots x_t \equiv AAACC$ cand = A count = 1 =>

* 1 occurence of A
* $(t-1)/2 = 2$ pairs (A, C) (A, C)

It is easy to see that Property P is maintained at every step. (exercise)

Let $cand_n$ and $count_n$ be the values of cand and count at the end of the stream and let m = true majority element of Σ

We now show that $cand_n = m$

By contradiction, suppose $cand_n \neq m$

Since Property P holds at the end of Σ, we have that Σ contains

* $count_n$ occurences of $cand_n$
* $(n - count_n)/2$ pairs (e_1, e_2) with $e_1 \neq e_2$

If $cand_n \neq m$ then m occurs in Σ at most $(n - count_n)/2$ times, and since $count_n \geq 0$ we have that

$(n - count_n)/2 \leq n/2$

and this contradicts the assumption that m is the majority element □

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 a random sample of the elements seen so far is required (e.g., *monitoring of sensor data*)

at every time step

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

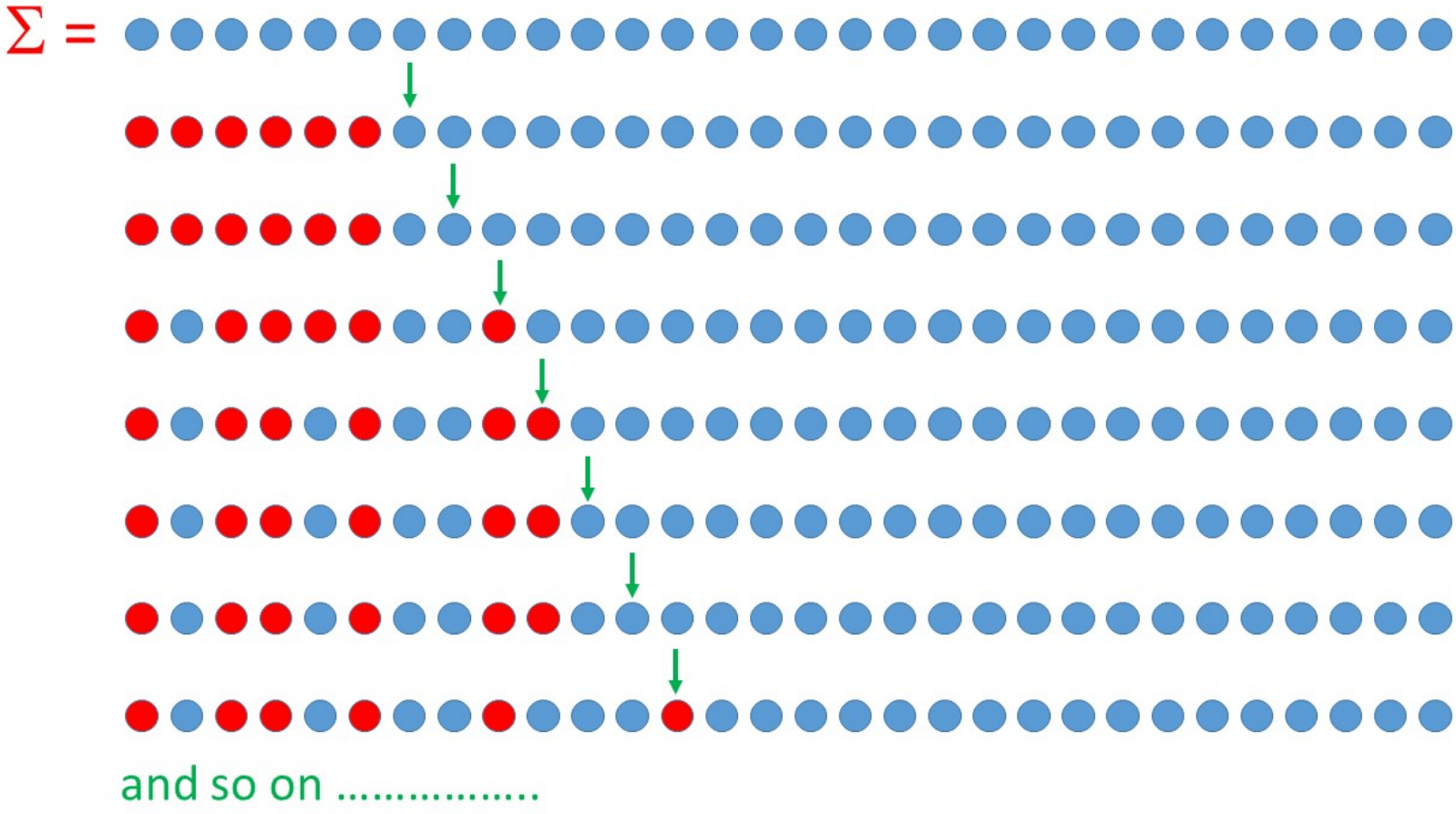
The following algorithm, dubbed Reservoir Sampling was developed by J.S. Vitter in 1985.

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

Proof

Let S_t = S after processing $x_1, x_2, \ldots, x_t$

We now show by induction on $t \geq m$ that S_t is an m-sample of $x_1 x_2 \ldots x_t$, that is, for any $i \in [1, t]$ $P_r(x_i \in S_t) = m/t$

BASE of INDUCTION $t = m$ straightforward

INDUCTIVE STEP Suppose that the property holds for S_{t-1}, for some $t-1 \geq m$. We now prove it for S_t

Inductive Hp.: $\forall i \in [1, t-1]$ $P_2(x_i \in S_{t-1}) = \frac{m}{t-1}$

* $P_2(x_t \in S_t) = m/t$ by construction
* Consider $i < t$ and define the event

$A =$ "x_t is added to S in step t"

Note that $P_2(A) = m/t$ and $P_2(\sim A) = 1 - m/t$

By the LAW of TOTAL PROBABILITIES we have

$$P_2(x_i \in S_t) = P_2((x_i \in S_t) \wedge A) + P_2((x_i \in S_t) \wedge \sim A)$$

(1) $P_2((x_i \in S_t) \wedge A) = P_2((x_i \in S_t) \mid A) * \underbrace{P_2(A)}_{= m/t}$

Now

$P_2((x_i \in S_t) \mid A) = \underbrace{P_2(x_i \in S_{t-1})}_{= m/(t-1)} * \underbrace{P_2(x_i \text{ not evicted in step } t)}_{= 1 - 1/m}$

by induction

by construction

$$\Rightarrow P_2((x_i \in S_t) \wedge A) = \frac{m}{t-1} \cdot \left(1 - \frac{1}{m}\right) \cdot \frac{m}{t}$$

(2) $P_2((x_i \in S_t) \wedge \sim A) = P_2((x_i \in S_t \mid \sim A) * \underbrace{P_2(\sim A)}_{= 1 - m/t}$

$$P_2\left((x_i \in S_t) \mid \sim A\right) = P_2\left(x_i \in S_{t-1}\right) = \frac{m}{t-1} \quad \text{by induction}$$

$$\Rightarrow P_2\left((x_i \in S_t) \wedge \sim A\right) = \frac{m}{t-1}\left(1 - \frac{m}{t}\right)$$

By combining (1) and (2) we get

$$P_2\left(x_i \in S_t\right) = \frac{m}{t-1}\left(1 - \frac{1}{m}\right)\frac{m}{t} + \frac{m}{t-1} \cdot \left(1 - \frac{m}{t}\right)$$

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

□

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).

Some observations

* There are at most $1/\varphi$ distinct frequent items in Σ since

  $n \geq (\#\text{frequent items}) * \varphi \cdot n$

* Any EXACT STRATEGY must keep a counter for every distinct item (even if not frequent)

  $\Rightarrow$ may require too much working memory up to $\Theta(n)$

* NEED to GIVE UP EXACTNESS

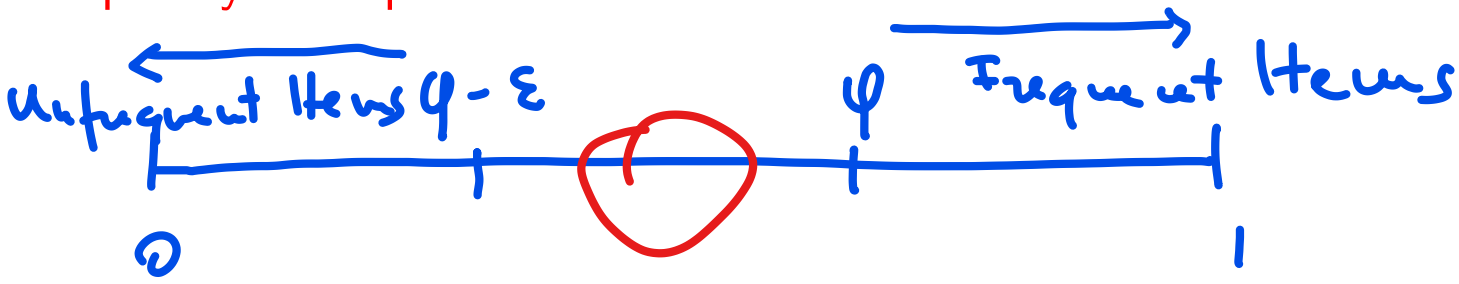
Approximate Frequent Items

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.



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.

Therefore: this approach does not properly solve the ϵ-AFI problem.

Let's argue why the first bullet is true

$a \equiv$ item occurring $n_a \geq \varphi n$ times in Σ

$m_a \equiv$ # occurrences of a in the sample S

Let $x_{i_1} = x_{i_2} = \cdots = x_{i_{n_a}} = a$ be the occurrences of a in Σ $\quad 1 \leq i_1 < i_2 < \cdots < i_{n_a} \leq n$

m_a is a random variable

$$E[m_a] = \sum_{j=1}^{n_a} P_2(x_{i_j} \in S) = n_a \cdot \frac{m}{n} \geq \varphi \cdot n \cdot \frac{m}{n}$$

$$= \varphi m$$

$\Rightarrow$ If $m \geq 1/q$ then $E[m_a] \geq 1$

Nevertheless, using Reservoir Sampling you may have false negatives and infrequent 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 $(x, f_e(x)) \in S$ **then** $f_e(x) \leftarrow f_e(x) + 1$;
else add $(x, 1)$ to S with probability r/n; /* (*start tracking* x) */

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 $O(\log(1/(\delta\varphi))/\epsilon)$, in expectation;*
- *1 pass;*
- *$O(1)$ expected processing time per element.*

Remark: The space is independent of n and constant for constant $\varphi, \epsilon, \delta$.

Note that if we assume φ and δ small enough to make $\ln(1/(\delta\varphi)) \geq 1$ ($\Rightarrow \delta\varphi \leq 1/e$), we have that since $\epsilon < \varphi$ then $\frac{1}{\epsilon}\ln\frac{1}{\delta\varphi} \geq \frac{1}{\varphi}$ which makes sense

Proof of Theorem

WORKING MEMORY

$Y_i \equiv$ Bernoulli variable $= \begin{cases} 1 & \text{if } x_i \text{ is a new entry in } S \\ 0 & \text{otherwise} \end{cases}$

Note that at the end of the algorithm

$$|S| = \sum_{i=1}^{n} Y_i \Rightarrow E[|S|] = E\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} E[Y_i]$$

$$E[Y_i] = P_r(Y_i = 1) \leq r/n$$

$$\Rightarrow E[|S|] \leq n \cdot r/n = r = \frac{1}{\varepsilon} \ln \frac{1}{\delta\varphi}$$

The working memory is dominated by S

1 PASS Straightforward

$O(1)$ EXPECTED TIME for ELEMENT: derives from the fact that we use a Hash Table as long as the number of cells in the Hash table is $\geq c \cdot |S|$ with c constant

CORRECTNESS

* By construction, Sticky Sampling cannot return an item that occurs less than $(\varphi - \varepsilon) \cdot n$ times in Σ

$\Rightarrow$ the second requirement in the specification of the ε-AFI problem is met!

* Now we are left to prove that, with probability $\geq 1-\delta$, no frequent items are missed

Let K = # frequent items in Σ

$\Rightarrow K \leq 1/\varphi$

Let y be an "arbitrary" frequent item

Crucial observation: if y is not in S at the end of the algorithm, then from none of the first $\lceil \varepsilon n \rceil$ occurrences of y in Σ have been sampled

$$P_2(y \text{ not returned}) \leq$$

$$P_2(\text{none of the first } \lceil \varepsilon n \rceil \text{ occurrences of } y \text{ is sampled}) \leq$$

$$\left(1 - \frac{r}{n}\right)^{\lceil \varepsilon n \rceil} \leq \left(1 - \frac{r}{n}\right)^{\varepsilon n} = \left(1 - \frac{r}{n}\right)^{\frac{n}{r} \cdot \frac{r}{n} \varepsilon n}$$

$$\leq \left(\frac{1}{e}\right)^{r \cdot \varepsilon}$$

we are using the inequality

$$(1 - 1/x)^x \leq 1/e \quad \forall\, x \geq 1$$

And $\left(\frac{1}{e}\right)^{r \cdot \varepsilon} = \left(\frac{1}{e}\right)^{\frac{1}{\varepsilon} \ln \frac{1}{\delta\varphi} \cdot \varepsilon} = \delta\varphi$

Let $a_1, a_2 \ldots a_k$ be the frequent items. By UNION BOUND we get that

$$\Pr(\exists\, a_i \text{ not returned}) \leq \sum_{i=1}^{k} \Pr(a_i \text{ not returned})$$

$$\leq k \cdot \delta\varphi \leq \delta \qquad (k \leq 1/\varphi)$$

$\Rightarrow \Pr(\text{all frequent items are returned}) \geq 1 - \delta$

□

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

Σ

Sampling probability for B_0 is 1

" " for B_i is $1/2^i$

RECALIBRATION: At the beginning of B_i assume that S is the same as it would be if sampling probability r/n

Sticky Sampling with unknown n

had been used from the beginning, from x_1 up to x_n, with $n = |B_0| + |B_1| + \cdots + |B_{i-1}| = 2^i r$

The purpose of recalibration at the beginning of B_i is to adjust S so to make it "look like" it was obtained using sampling probability $r/(2n)$ from the beginning

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 $\tau_x =$ number of tails before head of an unbiased coin.
    if $f_e(x) - \tau_x > 0$ **then** $f_e(x) = f_e(x) - \tau_x$
    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 $1/2^i$;

RECALIBRATION

Sticky Sampling with unknown n: analysis

After processing the prefix $\Sigma_n = x_1, x_2, \ldots, x_n$ the algorithm ensures the following properties. $n \geq 2r$

1. The current sampling rate is in $[r/n, 2r/n]$.
2. The probability that some given element a is in S is the same as the one we would get in the case the current sampling rate were used since the beginning. rewriten in Slide 52
3. The expected size of S is at most $2r = O\left(\log(1/(\delta\varphi))/\epsilon\right)$.
4. The set of items x in S with $f_e(x) \geq (\varphi - \epsilon)n$ is a solution to the ϵ-AFI problem for Σ_n, with probability $\geq 1 - \delta$.

①

Sampling probability $= \frac{1}{2^i}$

$$2^i r < n \leq 2^{i+1} \cdot r \Rightarrow x_n \in B_i$$

The sampling probability when x_n is processed is

$$\frac{r}{n} < \frac{r}{2^i \cdot r} = \frac{1}{2^i} = \frac{2r}{2^{i+1} \cdot r} \leq \frac{2r}{n}$$

We rewrite ② as follows

(2.1) For each i $1 \le i \le n$ the probability that x_i, when processed, is added to S as a new entry (not previously in S), is AT MOST $2r/n$

(2.2) If y occurs $\geq \varphi n$ times in $x_1, x_2 \ldots x_n$ then the first $\lceil \varepsilon n \rceil$ occurrences of y have been missed with probability $\leq \left(1 - \frac{r}{n}\right)^{\lceil \varepsilon n \rceil}$ as in the case of n Known

Using the same arguments employed in the analysis of the case n known we can prove

③ based on (2.1)

④ based on (2.2)

Sketching

Sketches

Sketch: space-efficient data structure that can be used to provide (typically probabilistic) estimates of (statistical) characteristics of a data stream.

Frequency Moments

Consider a stream $\Sigma = x_1, x_2, \ldots, x_n$, whose elements belong to a universe U.

For each $u \in U$ occurring in Σ define its (*absolute*) *frequency*

$$f_u = |\{j : x_j = u, 1 \leq j \leq n\}|,$$

i.e., the number of occurrences of u in Σ

Definition: Frequency Moments

For every $k \geq 0$, the k-th frequency moment F_k of Σ is

$$F_k = \sum_{u \in U} f_u^k. = \sum_{\substack{u \in U \\ u \text{ occurring in } \Sigma}} f_u^k$$

(We assume $0^0 = 0$)

Frequency Moments

Relevant informations from frequency moments:

- F_0 = number of distinct items in Σ. It is useful, for instance, in the analysis of web-logs.
- $F_1 = |\Sigma|$: trivial to maintain with a counter.
- $1 - F_2/|\Sigma|^2$ = Gini index of Σ. It provides info on data skew: the closer to 0 and the higher is the skew. It is used for decision tree induction.

- If an element occurs n times $\Rightarrow F_2 = n^2 = |\Sigma|^2$ and the GINI index is 0 (MAXIMUM SKEW)
- If m elements occur n/m times each, $F_2 = m\left(\frac{n}{m}\right)^2 = \frac{n^2}{m}$ and GINI index is $1 - 1/m$ (MAXIMUM BALANCE)

Computation of F_0 for Σ

or a dictionary with at most $|\Sigma|$ counters

Exact computation: use $|U|$ counters (not suited for streaming setting).

Approximation: Probabilistic Counting ([Flajolet, Martin 1983])

- Working memory $O(\log |U|)$.
- F_0 estimated within a factor c with probability $\geq 1 - 2/c$ (accuracy-confidence tradeoff).
- **Main idea:**
  - Map each $u \in U$ to a random integer $h(u) \in [0, |U| - 1]$.
    $\Rightarrow h(u)$ is a $O(\log |U|)$-bit binary string.
  - The more distinct elements in Σ, the more likely to have a $u \in \Sigma$ mapped to a string with many trailing 0's.

Probabilistic counting: details

We need the following ingredients:

- Array C of $\lceil \log(|U|+1) \rceil$ bits, all initialized to zero.
- Hash Function $h : U \to [0..|U|-1]$. We assume:
  - For every $u \in U$, $h(u)$ has uniform distribution in $[0..|U|-1]$
  - All $h(u)$'s are pairwise independent.
- Notation: for $i \in [0..|U|-1]$, define

  $t(i) =$ number of trailing zeroes in binary representation of i

  e.g., $12 = (1100)_2 \Rightarrow t(12) = 2$

Algorithm: **For each** $x_j \in \Sigma$ **do** $C[t(h(x_j)] \leftarrow 1$

After processing x_n, estimate F_0 as

$$\tilde{F}_0 = 2^R,$$

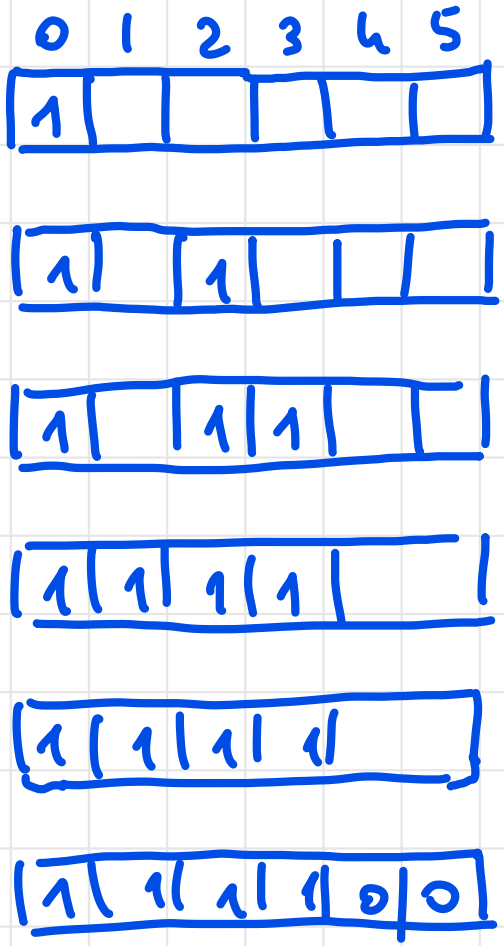
where R is the largest index of C with $C[R] = 1$.

Example

$\Sigma = A, D, A, A, C, B, F, F, B, A, E, C$ $\quad |U| = 26$

x	$h(x)$	$t(h(x))$
A	01001	0
B	111 00	2
C	11000	3
D	0111 0	1
E	10100	2
F	10011	0

C



$R = 3 \Rightarrow \tilde{F}_0 = 2^R = 8$ the true F_0 is 6

Analysis: intuition

For simplicity, assume $|U|$ a power of 2. Let $x \in \Sigma$.

What is the probability that $h(x)$ has at least j trailing zeros?

- $\text{Prob}(t(h(x)) \geq 1) = 1/2$
- $\text{Prob}(t(h(x)) \geq 2) = 1/4$

  .....
- $\text{Prob}(t(h(x)) \geq j) = 1/2^j$

Hence: In order to set a bit of index $\geq j$ to 1, we must process, $\Omega(2^j)$ distinct elements in expectation! This explains why 2^R is a good estimate for F_0.

Probabilistic guarantees

The following theorem summarizes the properties of the algorithm.

Theorem

For a stream Σ of n elements, the **probabilistic counting algorithm** *returns a value $\tilde{F}_0$ such that, for any $c > 2$:*

$$\Pr(\tilde{F}_0 < F_0/c) \leq 1/c \quad \textit{and} \quad \Pr(\tilde{F}_0 > cF_0) \leq 1/c,$$

hence

$$Pr(F_0/c \leq \tilde{F}_0 \leq cF_0) \geq 1 - 2/c.$$

The algorithm requires a working memory of $O(\log |U|)$ bits, 1 pass, and $O(\log |U|)$ time per element.

Exercises

Exercise: High Probability Guarantees (median trick)

Consider a stream Σ with elements from a universe U and let F_0 be the number of distinct elements in Σ. Suppose that you run ℓ *independent instances* of the probabilistic counting argument, obtaining ℓ estimates $\tilde{F}_0^{(j)}$ for F_0, with $1 \leq j \leq \ell$. Assume ℓ odd and let $\tilde{F}_0$ be the mediam value of the $\tilde{F}_0^{(j)}$'s. Show that with a suitable choice of $\ell = \Theta(\log |U|)$ we have:

$$\Pr(\tilde{F}_0 < F_0/16) \leq 1/|U| \quad \text{and} \quad \Pr(\tilde{F}_0 > 16F_0) \leq 1/|U|.$$

Hint: Use the previous theorem and the Chernoff bound.

To recap

- Streaming algorithms
  - Input received as a stream;
  - Small working memory and update time.
- Sampling
  - Reservoir sampling: to compute an uniform sample of a stream;
  - Sticky sampling: to find the frequent items in a stream.
- Sketching
  - A small summary of the data for estimating some characteristics of the stream (e.g., moments);
  - Probabilistic counter for counting the number of distinct elements.

References

- **[LRU14]** J. Leskovec, A. Rajaraman and J. Ullman. Mining Massive Datasets. Cambridge University Press, 2014. Chapter 4 (Section 4.1., 4.2, 4.4) (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)