Lecture 10

Data Stream Model: Frequency Moments

Course: Algorithms for Big Data

Instructor: Hossein Jowhari

Department of Computer Science and Statistics
Faculty of Mathematics
K. N. Toosi University of Technology

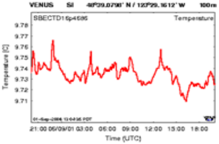
Spring 2021

Data Stream Model

Data stream: In data stream model, input data is presented to the algorithm as a stream of items in no particularly order. The data stream is read only once and cannot be stored (entirely) due to the large volume.

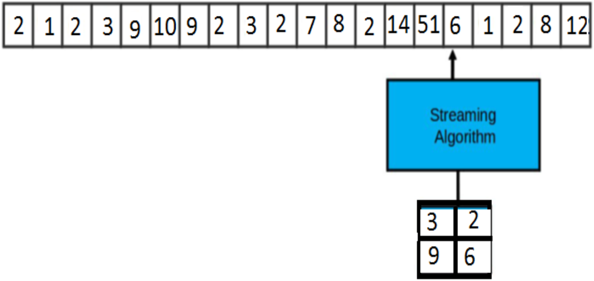
Examples of data streams:

- sensor data : temperature, pressure, ...
- website visits, click streams
- user queries (search)
- social network activities
- business transactions
- call center records



Data Stream Model

Streaming Algorithm is an algorithm that processes a data stream and has small memory compared with the amount of data it processes.



Sublinear space usage: Assuming n is the size of input typically a streaming algorithm has $o(n)$ (for example $\log^2(n)$, $\sqrt{n}$, etc) space usage.

Data Stream Model: two motivating puzzles

Missing elements in a permutation: Suppose the stream is a permutation of $\{1, \ldots, n\}$ with one element missing. How much space is needed to find the missing element?

$$7, 8, 3, 9, 1, 5, 2, 6, 10, 12, 11$$

What if 2 elements are missing?

Can we generalize to k missing elements?

Majority element: Suppose the stream is a sequence of numbers $a_1, \ldots, a_m$. Suppose one element is repeated at least $\frac{m}{2}$ times. How can we find the majority element?

$$2, 3, 2, 1, 2, 9, 8, 2, 5, 2, 2, 4, 6, 2, 2, 5, 2$$

Frequency Moments

Let $A = a_1, a_2, \ldots, a_m$ be the input stream where each $a_i \in \{1, \ldots, n\}$. Let x_i denote the number of repetitions of i in A. We define the k-th frequency moment of A

$$F_k = \sum_{i=1}^{n} x_i^k$$

Example: $A = 2, 3, 2, 1, 2, 9, 8, 2, 5, 2, 2, 4, 6, 2, 2, 5, 2$
$x_1 = 1, x_2 = 9, x_3 = 1, x_4 = 1, x_5 = 2, x_6 = 1, x_7 = 0, x_8 = 1, x_9 = 1$

$F_0 = \text{number of distinct elements } = 8$

$F_1 = \sum_{i=1}^{n} x_i = 17 = m$

$F_2 = \sum_{i=1}^{n} x_i^2 = 1^2 + 9^2 + 1^2 + 1^2 + 2^2 + 1^2 + 0^2 + 1^2 + 1^2 = 91$

$F_\infty = \max_{i=1}^{n} x_i$

Computing F_k in small space

Trivial facts:

- We can compute F_1 exactly in $O(1)$ words ($O(\log m)$ bits) of space.
- When k is a constant, we can compute F_k exactly in $O(n)$ words of space.

Nontrivial facts:

- Assuming $k \neq 1$, any randomized streaming algorithm that computes F_k exactly requires $\Omega(n)$ space.
- Assuming $k \neq 1$, any deterministic streaming algorithm that computes a constant factor approximation of F_k requires $\Omega(n)$ space.
- Both randomization and approximation is needed to compute F_k in sublinear space.

Approximating F_2

The space complexity of approximating the frequency moments *

Noga Alon † Yossi Matias ‡ Mario Szegedy §

February 22, 2002

Abstract

The frequency moments of a sequence containing m_i elements of type i, for $1 \leq i \leq n$, are the numbers $F_k = \sum_{i=1}^{n} m_i^k$. We consider the space complexity of randomized algorithms that approximate the numbers F_k, when the elements of the sequence are given one by one and cannot be stored. Surprisingly, it turns out that the numbers F_0, F_1 and F_2 can be approximated in logarithmic space, whereas the approximation of F_k for $k \geq 6$ requires $n^{\Omega(1)}$ space. Applications to data bases are mentioned as well.

Theorem [AMS99] There is a randomized streaming algorithm that approximates F_2 within $1 + \epsilon$ factor using $O(\frac{1}{\epsilon^2})$ words of space. The algorithm succeeds with probability $3/4$.

JL Lemma and Approximating F_2

$$A = 2, 3, 2, 1, 2, 9, 8, 2, 5, 2, 2, 4, 6, 2, 2, 5, 2$$

$$\boldsymbol{x} = \begin{array}{ccccccccc} m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 & m_8 & m_9 \\ \hline \boxed{1} & \boxed{9} & \boxed{1} & \boxed{1} & \boxed{2} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{1} \end{array}$$

$$F_2 = \|\boldsymbol{x}\|^2$$

> Consequence of Lemma 2, previous lecture: Given $\boldsymbol{x} \in \mathbb{R}^n$, assuming $t \geq \frac{c}{\epsilon^2}$ when c is a large enough constant then we have
>
> $$Pr\left(\|f(\boldsymbol{x})\|^2 \approx_\epsilon \|\boldsymbol{x}\|^2\right) \geq 3/4$$

Assume we have $\boldsymbol{x}$ and $\boldsymbol{M}$. We can compute an approximation of $F_2 = \|\boldsymbol{x}\|^2$ by computing $\|f(x)\|^2$.

$$\frac{1}{\sqrt{3}} \overbrace{\begin{bmatrix} -0.23 & -0.02 & -0.22 & -0.68 & +0.39 & +0.24 & +0.36 & +0.47 & -1.42 \\ +0.08 & +0.46 & +0.68 & +0.47 & -0.28 & +1.90 & +1.13 & -1.09 & +2.27 \\ +0.89 & -0.24 & +0.83 & +1.92 & -0.47 & +0.10 & +0.33 & -0.90 & -0.99 \end{bmatrix}}^{\boldsymbol{M}} \overbrace{\begin{bmatrix} 1 \\ 9 \\ 1 \\ 1 \\ 2 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}}^{\boldsymbol{x}} = \overbrace{\begin{bmatrix} -1.24 \\ 7.89 \\ 1.92 \end{bmatrix}}^{f(\boldsymbol{x})}$$

But if we store $\boldsymbol{M}$ and $\boldsymbol{x}$, it would take a lot of space.

For now, lets assume we have stored $\boldsymbol{M}$ (later we remove this assumption.) We show how $f(\boldsymbol{x})$ is computed from the stream A.

Every item in the stream is a single update of the vector $\boldsymbol{x}$ (it increments one of its coordinates) In the beginning, $\boldsymbol{x} = \mathbf{0}$ and $f(\boldsymbol{x}) = \mathbf{0}$.

Lets see when the i-th coordinate of $\boldsymbol{x}$ is incremented, how does $f(\boldsymbol{x})$ change?

Let $f(\boldsymbol{x})_j$ be the j-th coordinate of $f(\boldsymbol{x})$. $f(\boldsymbol{x})_j$ is a inner product of $\boldsymbol{x}$ and the j-th row of $\boldsymbol{M}$ (divided by $\sqrt{t}$). When x_i is increased by 1, we need to add $\frac{1}{\sqrt{t}} M_{ji}$ to $f(\boldsymbol{x})_j$.

$$\text{increment } x_i \;\Rightarrow\; \text{add } \; \frac{1}{\sqrt{t}} M_{ji} \; \text{ to } \; f(\boldsymbol{x})_j$$

How the stream updates x

$A = 2, 3, 2, 1, 2, 9, 8, 2, 5, 2, 2, 4, 6, 2, 2, 5, 2$

Before the stream arrives

x =

0	0	0	0	0	0	0	0	0

$A = 2, 3, 2, 1, 2, 9, 8, 2, 5, 2, 2, 4, 6, 2, 2, 5, 2$

Somewhere in the middle of the stream

x =

1	4	1	0	0	0	0	1	1

$A = 2, 3, 2, 1, 2, 9, 8, 2, 5, 2, 2, 4, 6, 2, 2, 5, 2$

After the next item

x =

1	4	1	0	1	0	0	1	1

$A = 2, 3, 2, 1, 2, 9, 8, 2, 5, 2, 2, 4, 6, 2, 2, 5, 2$

At the end of the stream

x =

1	9	1	1	2	1	0	1	1

So if we have $\boldsymbol{M}$, computing $f(\boldsymbol{x})$ from the stream is straightforward. We only need to store $\boldsymbol{M}$ and $f(\boldsymbol{x})$.

The vector $f(\boldsymbol{x})$ has t coordinates and therefore we need only $O(t)$ words to maintain it.

The coefficient matrix $\boldsymbol{M}$ is generated in the beginning but we need to store it as the stream arrives. This takes $O(nt)$ words of space. ☹

Recall in the end of last lecture, we mentioned that the Gaussian coefficients can be replaced by $\{-1, +1\}$ random numbers. This not only makes the algorithm easier to implement but also helps us in getting rid of the need to store $\boldsymbol{M}$. But how?

AMS's paper uses $\{-1, +1\}$ random coefficients instead of Gaussian random numbers.

For each coordinate of $\boldsymbol{x}$, we pick a random number σ_i from $\{-1, +1\}$. For now, suppose we have stored the vector $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$.

Let $Z = \sum_{i=1}^{n} \sigma_i x_i$. Z is the inner product of $\boldsymbol{\sigma}$ and $\boldsymbol{x}$. Note that Z is gradually computed as the input stream arrives. We analyze $X = Z^2$.

$$E[X] = E[(\sum_{i=1}^{n} \sigma_i x_i)^2] = \sum_{i=1}^{n} E[\sigma_i^2] x_i^2 + \sum_{i \neq j} E[\sigma_i \sigma_j] x_i x_j$$

Because σ_i and σ_j are independent, we get

$$E[X] = \sum_{i=1}^{n} E[\sigma_i^2] x_i^2 + \sum_{i \neq j} E[\sigma_i] E[\sigma_j] x_i x_j$$

Because $E[\sigma_i] = 0$ and $E[\sigma_i^2] = 1$, we get

$$E[X] = \sum_{i=1}^{n} x_i^2 = F_2$$

Using Chebyshev Inequality, we can say

$$Pr(|X - E[X]| > \epsilon E[X]) = \frac{Var[X]}{\epsilon^2 E^2[X]}$$

Because σ_i's are independent,

$$E[X^2] = E[Z^4] = E[(\sum_{i=1}^{n} \sigma_i x_i)^4] =$$

$$\sum_{i=1}^{n} E[\sigma_i^4]x_i^4 + 6\sum_{i,j} E[\sigma_i^2\sigma_j^2]x_i^2x_j^2 + 4\sum_{i,j} E[\sigma_i\sigma_j^3]x_ix_j^3$$

$$= \sum_{i=1}^{n} x_i^4 + 6\sum_{i,j} x_i^2x_j^2$$

$$Var[X] = E[X^2] - E^2[X] \leq 4\sum_{i,j} x_i^2x_j^2 \leq 2F_2^2$$

Consequently,

$$Pr(|X - E[X]| > \epsilon E[X]) = \frac{Var[X]}{\epsilon^2 E^2[X]} \leq \frac{2F_2^2}{\epsilon^2 F_2^2} = \frac{2}{\epsilon^2}$$

Repeat to Decrease the Variance

To decrease the variance of X, we compute $s = \frac{8}{\epsilon^2}$ independent copies $Y_1, \ldots, Y_s$ of X and output their average $Y = \frac{1}{s}(Y_1 + \ldots + Y_s)$

Note that $E[Y] = E[X] = F_2$ and $Var[Y] = \frac{1}{s}Var[X]$

$$Pr(|Y - E[Y]| \geq \epsilon E[Y]) \leq \frac{Var[Y]}{\epsilon^2 E^2[Y]} \leq \frac{1}{4}$$

$$Pr(|Y - F_2| \geq \epsilon F_2) \leq \frac{Var[Y]}{\epsilon^2 E^2[Y]} \leq \frac{1}{4}$$

What about the space needed to store $\boldsymbol{\sigma}$?

We do not need to store the random vector $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$. Why?

Notice, it is enough the random coefficients σ_i's to be 4-wise independent. We do not need them to be totally independent! See the analysis of $E[X]$ and $E[X^2]$.

We can generate a set of n k-wise independent random numbers using $O(k \log n)$ random bits.

How? This is the topic of the next lecture.