Algorithms for Big Data (Fall 2020)

Instructor: Chihao Zhang
Scribed by: Guoliang Qiu

September 23, 2020

Designing efficient algorithms for large data sets poses challenges for traditional algorithmic techniques in many aspects. This course will mainly focus on three topics developed in processing big data, including 1) sublinear space algorithms, 2) almost linear-time algorithms, and 3) online algorithms. In the first part of this course, I will introduce the "streaming model" and related algorithmic design technique. They are quite different from the algorithms you might have learnt in the algorithm course and are quite useful for tackling problems with super large input data.

1 Introduction to the Streaming Model

1.1 A Programmer for Routers

Suppose we have a router with limited memory, but need to solve some computational tasks with large input data such as monitoring the id of devices connecting to it. We can ask the following three natural questions in this scenario,

- How many numbers in a given data streaming?
- How many distinct numbers?
- What is the most frequent number?

In order to study this problem systematically, we first introduce the streaming model and formally define the computational problem we are considering in the model.

1.2 Streaming Model

In the streaming model, the input is a sequence $\sigma = \langle a_1, a_2, \cdots, a_m \rangle$ where each $a_i \in [n]$. We should notice that the data is coming one by one which explains where the name streaming comes from. Assume one can process the input stream using at most s bits of memory. We say an algorithm is sublinear if $s = o(\min\{m, n\})$. The computational tasks arisen from the streaming model include the following,

- How many numbers (What is m?)
- How many distinct numbers in σ?
- What is the median of σ?
- What is the most frequent number?

- ...

In today's lecture, we focus on the first question, i.e., can we design a sublinear algorithm to compute m?

2 Counting the Elements in Streaming Model

2.1 A Naive Idea

It is easy to obtain a simple algorithm to compute m. We can maintain a counter k, and whenever one reads a number a_i, let $k = k + 1$. It is not hard to see that we need $\lceil \log_2 m \rceil$ memory with this naive algorithm.

Certainly, we can ask that whether we can design a clever algorithm with only $o(\log m)$ memory? It turns out that computing the exact answer is impossible even with $\lceil \log_2 m \rceil - 1$ memory. The reason is as follows: suppose we have an algorithm $\mathcal{M}$ using only $\lceil \log_2 m \rceil - 1$ memory and denote $\mathcal{M}(i)$ as the output of the algorithm with a input σ of length i, then there exists $i, j \in \{m\}$ such that $i \neq j$ and $\mathcal{M}(i) = \mathcal{M}(j)$.

Even though we can not get a better algorithm in the exact regime, it is possible to obtain a more efficient algorithm if we allow approximation, i.e., we want to design an algorithm such that for every $\varepsilon > 0$, compute a number $\widehat{m}$ such that

$$1 - \varepsilon \leq \frac{\widehat{m}}{m} \leq 1 + \varepsilon.$$

2.2 Morris' Algorithm

The Morris' algorithm is presented as Algorithms 1, we know that it is a randomized algorithm. Therefore we look at the expectation of its output.

Algorithm 1 Morris' Algorithms for Counting Elements

Input: *An instance* $\sigma = \langle a_1, a_2, \cdots, a_m \rangle$ *where each* $a_i \in [n]$.
Output: *The length* m *of the sequence* σ.
1: **Init**: A varaible $X \leftarrow 0$
2: **On Input y**: Increase X with probability 2^{-X}
3: **Output**: Output $\widehat{m} = 2^X - 1$.

Lemma 1. *The output of Morris' algorithm* $\widehat{m}$ *satisfies that its expectation* $\mathbf{E}[\widehat{m}] = m$.

Proof. We prove it by induction on m. Since $X = 1$ when $m = 1$, we have $\mathbf{E}[\widehat{m}] = 1$. Assume it is true for

smaller m, let X_i denote the value of X after processing i-th input. We have the following fact,

$$\begin{aligned}
\mathbf{E}\left[\widehat{m}\right] &= \mathbf{E}\left[2^{X_m}\right] - 1 && (1)\\
&= \sum_{i=0}^{m} \mathbf{Pr}\left[X_m = i\right] \cdot 2^i - 1 && (2)\\
&= \sum_{i=0}^{m} \left(\mathbf{Pr}\left[X_{m-1} = i\right] \cdot \left(1 - 2^{-i}\right) + \mathbf{Pr}\left[X_{m-1} = i-1\right] \cdot 2^{1-i}\right) \cdot 2^i - 1 && (3)\\
&= \sum_{i=0}^{m-1} \mathbf{Pr}\left[X_{m-1} = i\right] \cdot (2^i + 1) - 1 && (4)\\
&= \mathbf{E}\left[2^{X_{m-1}}\right] && (5)\\
&= m && (6)
\end{aligned}$$

where equation 6 holds due to the induction hypothesis.

□

It it now clear that Morris' algorithm is an unbiased estimator for m and uses approximately $O(\log \log m)$ bits of memory. However, for a practical randomized algorithm, we further require its output to concentrate on the expectation. That is, we want to establish concentration inequality of the form

$$\mathbf{Pr}\left[|\widehat{m} - m| > \varepsilon\right] \leq \delta$$

for $\varepsilon, \delta > 0$. It is natural to see that for fixed ε, the smaller δ is, the better the algorithm will be.

3 Concentration Analysis of Morris' Algorithm

We need more probabilistic tools to establish the concentration inequalities.

Theorem 2 (Markov's inequality). *For every nonnegative random variable X and every $a \geq 0$, it holds that*

$$\mathbf{Pr}\left[X \geq a\right] \leq \frac{\mathbf{E}\left[X\right]}{a}$$

Theorem 3 (Chebyshev's inequality). *For every random variable X and every $a \geq 0$, it holds that*

$$\mathbf{Pr}\left[|X - \mathbf{E}\left[X\right]| \geq a\right] \leq \frac{\mathbf{Var}\left[X\right]}{a^2}$$

In order to apply Chebyshev's inequality to analyze the Morris' algorithm, we have to first compute the variance of $\widehat{m}$.

Lemma 4.

$$\mathbf{E}\left[\left(2^{X_m}\right)^2\right] = \frac{3}{2}m^2 + \frac{3}{2}m + 1$$

Proof. We can prove the claim using an induction argument similar to our proof for the expectation. When $m = 1$, $\mathbf{E}\left[\left(2^{X_m}\right)^2\right] = 4$. We assume it is true for smaller m and use the same notation X_i. We have that

$$\begin{aligned}
\mathbf{E}[\widehat{m}] &= \mathbf{E}\left[2^{X_m}\right] - 1 && (7)\\
&= \sum_{i=0}^{m} \mathbf{Pr}[X_m = i] \cdot 2^{2i} && (8)\\
&= \sum_{i=0}^{m} (\mathbf{Pr}[X_{m-1} = i]\,(-2^{-i}) + \mathbf{Pr}[X_{m-1} = i-1] \cdot 2^{1-i}) \cdot 2^{2i} && (9)\\
&= \sum_{i=0}^{m} (\mathbf{Pr}[X_{m-1} = i]\,(2^{2i} - 2^{i}) + \mathbf{Pr}[X_{m-1} = i-1] \cdot 2^{i+1}) && (10)\\
&= \sum_{i=0}^{m-1} \mathbf{Pr}[X_{m-1} = i]\,(2^{2i} + 3 \cdot 2^{i}) && (11)\\
&= \mathbf{E}\left[(2^{X_{m-1}})^2\right] + 3\mathbf{E}\left[2^{X_{m-1}}\right] && (12)\\
&= \frac{3}{2}m^2 + \frac{3}{2}m + 1 && (13)
\end{aligned}$$

□

With Lemma 4, we can compute the variance as follows,

$$\mathbf{Var}[\widehat{m}] = \mathbf{E}\left[\widehat{m}^2\right] - \mathbf{E}\left[\widehat{m}^2\right] = \mathbf{E}\left[\left(2^{X_m} - 1\right)^2\right] - m^2 \leq \frac{m^2}{2}$$

Applying Chebyshev's inequality, we obtain for every $\varepsilon > 0$,

$$\mathbf{Pr}[|\widehat{m} - m| \geq \varepsilon m] \leq \frac{1}{2\varepsilon^2}$$

However, we can observe that as ε becomes smaller and smaller, the above bound is not useful. Thus, it is necessary to ask that whether we can improve the concentration inequality? The answer is affirmative and there are two common tricks work here.

4 Tricks for Boosting the Concentration

4.1 Averaging Trick

The Chebyshev's inequality tells us that we can improve the concentration by reducing the variance. Thus, it is necessary to review some properties the variance satisfied first.

- $\mathbf{Var}[a \cdot X] = a^2 \cdot \mathbf{Var}[X]$
- $\mathbf{Var}[X + Y] = \mathbf{Var}[X] + \mathbf{Var}[Y]$ for mutually independent random variables X and Y.

We can design a new algorithm by independently running Morris' algorithm t time in parallel, and denote the corresponding outputs be $\widehat{m}_1, \cdots, \widehat{m}_t$. Then the final output is $\widehat{m}^* := \frac{\sum_{i=1}^{t} \widehat{m}_i}{t}$.

Applying Chebyshev's inequality to $\widehat{m}^*$:

$$\mathbf{Pr}\left[|\widehat{m}^* - m| \geq \varepsilon m\right] \leq \frac{1}{t \cdot 2\varepsilon^2}$$

For $t \geq \frac{1}{2\varepsilon^2\delta}$, we have

$$\mathbf{Pr}\left[|\widehat{m}^* - m| \geq \varepsilon m\right] \leq \delta$$

The new algorithm uses $O\left(\frac{\log\log n}{\varepsilon^2\delta}\right)$ bits of memory. It shows a trade-off between the quality of the randomized algorithm and the consumption of memory space.

4.2 The Median Trick

We can boost the performance furthermore by the median trick.

We choose $t = \frac{3}{2\varepsilon^2}$ in the previous algorithm after applying the averaging trick and independently run it s time in parallel. Denote the outputs as $\widehat{m}_1^*, \widehat{m}_2^*, \cdots, \widehat{m}_s^*$ respectively.

It holds that for every $i = 1, \cdots, s$,

$$\mathbf{Pr}\left[\left|\widehat{m}_i^* - m\right| \geq \varepsilon m\right] \leq \frac{1}{3}.$$

At last, we output the median $\widehat{m}^{**}$ of $\widehat{m}_1^*, \widehat{m}_2^*, \cdots, \widehat{m}_s^*$. To analyze the performance of the new algorithm, we use the Chernoff bound.

Theorem 5 (Chernoff Bound). *Let $X_1, \cdots, X_n$ be independent random variables with $X_i \in \{0, 1\}$ for every $i = 1, \cdots, n$. Let $X = \sum_{i=1}^n X_i$. Then for every $0 < \varepsilon < 1$, it holds that*

$$\mathbf{Pr}\left[|X - \mathbf{E}[X]| > \varepsilon \cdot \mathbf{E}[X]\right] \leq 2\exp\left(-\frac{\varepsilon^2\mathbf{E}[X]}{3}\right)$$

Then we can apply the Chernoff bound to analyze the result obtained by the Median trick. For every $i = 1, \cdots, s$, we let Y_i be the indicator of the (good) event

$$\left|\widehat{m}_i^* - m\right| < \varepsilon \cdot m.$$

Then $Y \triangleq \sum_{i=1}^s Y_i$ satisfies $\mathbf{E}[Y] \geq \frac{2}{3}s$. If the median $\widehat{m}^{**}$ is bad (namely $|\widehat{m}^{**} - m| \geq \varepsilon \cdot m$), then at least half of $\widehat{m}^*$'s are bad. Equivalently, $Y \leq \frac{1}{2}s$. By Chernoff bound,

$$\mathbf{Pr}\left[|Y - \mathbf{E}[Y]| \geq \frac{1}{6}s\right] \leq 2\exp\left(-\frac{s}{72}\right)$$

Therefore, for $t = O\left(\frac{1}{\varepsilon^2}\right)$ and $s = O\left(\log\frac{1}{\delta}\right)$, we have

$$\mathbf{Pr}\left[|\widehat{m}^{**} - m| \geq \varepsilon m\right] \leq \delta$$

This new algorithm use $O(\frac{1}{\varepsilon^2} \cdot \log\frac{1}{\delta} \cdot \log\log n)$ bits of memory.