Algorithms for Big Data (V) (Fall 2020)

Instructor: Chihao Zhang
Scribed by: Guoliang Qiu

Last modified on Nov 2, 2020

Consider a stream of numbers of $\sigma = \langle a_1, \cdots, a_m \rangle$ where $a_i \in [n]$ and its frequency vector $\mathbf{f} = (f_1, \cdots, f_n)$. In previous lectures, we learnt algorithms for following tasks:

- Counting the number of elements in the stream σ, i.e., $\|\mathbf{f}\|_1$.
- Counting the number of distinct elements in the stream σ, i.e., $\|\mathbf{f}\|_0$.
- Estimate the frequency f_a of a data $a \in [n]$. This implies that we can estimate $\|\mathbf{f}\|_\infty$.

Today we will introduce the AMS estimator which can estiamte all $\|\mathbf{f}\|_k$ for $k \geq 1$. At last, we will start the topic of graph streams.

1 The "Reservoir Sampling" Algorithm

Before showing how to compute the norm of $\mathbf{f}$, we first consider the following problem: how to uniformly sample $a_i \in [n]$ in a stream $\langle a_1, \cdots, a_m \rangle$ efficiently? This is a computational task which has been widely used in streaming algorithms. It can be done by using $O(n \cdot \log m)$ space trivially. A interesting trick called the "Reservoir Sampling" algorithm described below can reduce the cost to $O(\log m + \log n)$.

Algorithm 1 The Reservoir Sampling Algorithm

Init:
$x = 0, r = 0$.
On Input y:
$x \leftarrow x + 1$.
With probability $\frac{1}{x}$, $r \leftarrow x$.
Output:
Output r.

Lemma 1. *For any $i \in [m]$, $\mathbf{Pr}[r = i] = \frac{1}{m}$ in the Reservoir Sampling algorithm.*

Proof. We prove it by induction on m. For the base case $m = 1$, the algorithm simply outputs 1. Now assume $m > 2$ and the lemma holds for smaller m. Upon receiving a_m, we update $r \leftarrow m$ with probability $\frac{1}{m}$. So $\mathbf{Pr}[r = m] = \frac{1}{m}$. For other $k < m$, the algorithm outputs $r = k$ if and only if $r = k$ before receiving a_m and r is not updated. By induction hypothesis, the probability is $\mathbf{Pr}[r = i] = \frac{1}{m-1} \cdot \frac{m-1}{m} = \frac{1}{m}$. □

Furthermore, we can get the following corollary,

Corollary 2. *In the Reservoir Sampling algorithm, if we introduce variable z and update $z = y$ when updating r, then for any $a \in [n]$, $\mathbf{Pr}[z = a] = \frac{f_a}{m}$.*

2 AMS Estimator for F_k

In this section, we introduce the AMS algorithm to estimate $F_k := \sum_{a \in [n]} f_a^k$. The high level overview of the algorithm is the following 3 steps,

1. Pick $J \in [m]$ uniformly at random using the Reservoir Sampling algorithm,
2. Compute the number of elements equal to a_J after the J-th position, i.e., $r := \left|\left\{j \geq J : a_j = a_J\right\}\right|$,
3. Output $m(r^k - (r-1)^k)$.

The AMS estimator for F_k is described in Algorithm 2,

Algorithm 2 The AMS Estimator for F_k

Init:
$m \leftarrow 0, a \leftarrow 0, r \leftarrow 0$.
On Input y:
$m \leftarrow m + 1$.
With probability $\frac{1}{m}$, $a \leftarrow y, r \leftarrow 0$.
if $y = a$ **then** $r \leftarrow r + 1$.
end if
Output:
$\hat{F}_k = m \cdot (r^k - (r-1)^k)$.

The algorithm consumes $O(\log m + \log n)$ bits of memory. Now we shall prove that the estimate $\hat{F}_k$ is close to F_k with high probability. The expectation of $\hat{F}_k$ is

$$\begin{aligned}
\mathbf{E}\left[\hat{F}_k\right] &= \sum_{x \in [n]} \mathbf{Pr}[a = x] \cdot \mathbf{E}\left[\hat{F}_k \,\middle|\, a = x\right] \\
&= \sum_{x \in [n]} \frac{f_x}{m} \cdot m \cdot \mathbf{E}\left[r^k - (r-1)^k \,\middle|\, a = x\right] \text{ (Corollary 2)} \\
&= \sum_{x \in [n]} \frac{f_x}{m} \cdot m \cdot \sum_{r=1}^{f_x} \frac{1}{f_x} \left(r^k - (r-1)^k\right) \\
&= \sum_{x \in [n]} \frac{f_x}{m} \cdot m \cdot f_x^{k-1} \\
&= \sum_{x \in [n]} f_x^k.
\end{aligned}$$

The variance of $\hat{F}_k$ is

$$\begin{aligned}
\mathbf{Var}\left[\hat{F}_k\right] &\le \mathbf{E}\left[\hat{F}_k^2\right] = \sum_{x\in[n]} \mathbf{Pr}\left[a = x\right] \cdot \mathbf{E}\left[\hat{F}_k^2 | a = x\right] \\
&= \sum_{x\in[n]} \frac{f_x}{m} \cdot m^2 \cdot \sum_{r=1}^{f_x} \frac{1}{f_x} \cdot (r^k - (r-1)^k) \cdot (r^k - (r-1)^k) \\
&\le \sum_{x\in[n]} m \cdot f_x \sum_{r=1}^{k} \frac{1}{f_x} \cdot k \cdot r^{k-1} (r^k - (r-1)^k) \quad \text{(Mean-Value theorem)} \\
&\le \sum_{x\in[n]} m \cdot k \cdot f_x^{2k-1} = k \left(\sum_{x\in[n]} f_x\right)\left(\sum_{x\in[n]} f_x^{2k-1}\right).
\end{aligned}$$

In the above equation, $\left(\sum_{x\in[n]} f_x\right) = F_1$ and $\left(\sum_{x\in[n]} f_x^{2k-1}\right) = F_{2k-1}$. In order to use the Chebyshev inequality, we aim to compare $F_1 F_{2k-1}$ with F_k^2. First, we denote $a_* := \arg\max_x f_x$,

$$\begin{aligned}
\left(\sum_{x\in[n]} f_x\right)\left(\sum_{x\in[n]} f_x^{2k-1}\right) &\le \left(\sum_{x\in[n]} f_x\right) \cdot f_{a_*}^{k-1} \left(\sum_{x\in[n]} f_x^k\right) \\
&\overset{(\spadesuit)}{\le} \sum_{x\in[n]} \left(f_x^k\right)^{\frac{1}{k}} \cdot \left(\sum_{x\in[n]} f_x^k\right)^{\frac{k-1}{k}} \cdot F_k \\
&\overset{(\heartsuit)}{\le} n^{1-\frac{1}{k}} \cdot F_k^2,
\end{aligned}$$

where $(\spadesuit)$ comes from the fact that $f_{a_*}^{k-1} = \left(f_{a_*}^k\right)^{\frac{k-1}{k}} \le \left(\sum_{x\in[n]} f_x^k\right)^{\frac{k-1}{k}}$ and $(\heartsuit)$ follows from Jensen's inequality:

$$n \cdot \sum_{x\in[n]} \frac{1}{n}\left(f_x^k\right)^{\frac{1}{k}} \le n \cdot \left(\sum_{x\in[n]} \frac{1}{n} f_x^k\right)^{\frac{1}{k}} = n^{1-\frac{1}{k}} \cdot F_k^{\frac{1}{k}}.$$

In conclusion, we have $\mathbf{Var}\left[\hat{F}_k\right] \le k n^{1-\frac{1}{k}} F_k^2$. Applying Chebyshev's inequality gives

$$\mathbf{Pr}\left[\left|\hat{F}_k - F_k\right| \ge \epsilon F_k\right] \le \frac{k n^{1-\frac{1}{k}}}{\epsilon^2}.$$

Using the Average and Median tricks, we can boost the performance of AMS such that

$$\mathbf{Pr}\left[\left|\hat{F}_k - F_k\right| \ge \epsilon F_k\right] \le \delta,$$

with $O\left(\frac{1}{\epsilon^2} \log \frac{1}{\delta} k n^{1-\frac{1}{k}} (\log m + \log n)\right)$ bits of memory.

3 Graph Stream

In this section, we will introduce the graph streaming model. Suppose we have a graph with n vertices whose edges are unknown. A sequence of updates on the edge set comes in a streaming fashion. That is,

each time we receive an update of an edge $(u, v, +\backslash -)$ and may need to answer a certain query about some graph property on the graph in hand. If one stores all the edges of the graph, it cost $O(n^2)$ spaces in the worst case. Therefore, it is natural to ask whether one can correctly answer queries of graph properties using less memory. It seems that for many basic graph properties including the connectedness of the graph, $\Omega(n)$ space is necessary (and this will be rigorously proved in the class a few weeks later!). So in this case we no longer expect sublinear space algorithms.

3.1 Connectedness

First let's look at the connectedness of a graph. To determine whether a graph is connected, we only need to maintain a spanning forest F of it. The details are described in Algorithm 3.

Algorithm 3 The Connectedness Algorithm

Init:
$F \leftarrow \emptyset$, `flag` $\leftarrow 0$.
On Input $\{u, v\}$:
if `flag` $= 0$ and $F \cup \{\{u, v\}\}$ has no cycle **then**
  $F \leftarrow F \cup \{(u, v)\}$;
  if $|F| = n - 1$ **then** `flag` $\leftarrow 1$
  end if
end if
Output:
Output `flag`.

It is clear that the algorithm uses $O(n \log n)$ memory since at most the names of $n - 1$ edges need to be stored. Its correctness is also clear and thus we omit the proof.

3.2 Bipartiteness

A similar problem is to determine whether the graph is bipartite. We still maintain a spanning forest of the graph and test whether there are odd cycles. The algorithm is described in Algorithm 4.

We shall prove the correctness of the algorithm . The key point is that a graph is bipartite if and only if it has no odd cycles. If `flag` $= 0$, we know that there must exist an odd cycle in G, therefore it is non-bipartite. Otherwise, if `flag` $= 1$, there are no odd cycles in F. This means that there exists a 2-proper coloring of F, i.e., one can color the vertices with two colors so that the two ends of each edge in F receive different color. The 2-proper coloring can be extended to the whole G. To see this, we only need to check that the ends of those edges in $E(G) \setminus F$ are not monochromatic. In fact, for each $e = \{u, v\} \in E(G) \setminus F$, $F \cup e$ must contain an even cycle. So u and v are in different colors.

Algorithm 4 The Bipartiteness Testing Algorithm

Init:
$F \leftarrow \emptyset$, $\texttt{flag} \leftarrow 1$.
On Input $\{u, v\}$:
if $\texttt{flag} = 1$ **then**
  if $F \cup \{\{u, v\}\}$ has no cycle **then**
    $F \leftarrow F \cup \{(u, v)\}$;
  else
    if $F \cup \{(u, v)\}$ has an odd cycle **then** $\texttt{flag} \leftarrow 0$
    end if
  end if
end if
Output:
Output flag.