9 Streaming Algorithms

We can imagine a situtation in which a stream of data is being recieved but there is too much data coming in to store all of it. However, we want to extract some information out of the stream of data without storing all of it. An example could be a company like Facebook or Google storing click data or login data. What they really care about is aggregate data and not individual datum.

The following are all examples of streaming problems that we might be interested in. Some you will find have very simple algorithms and some are rather complicated. As an aside, it should be noted that the majority of work in streaming algorithms requires using randomized algorithms (which you can learn all about in CS/CMS 139). Typically, this is done by selecting a small sample of the data and estimating the aggregate data from the small sample.

- Calculating the length of the stream.
- Calculating the mean or median of a set of values.
- Calculating the standard deviation of a set of values.
- Calculating the number of unique elements (assuming the stream is elements from $\{1, \ldots, n\}$).[1]
- Calculating the k most frequent elements.[2]
- Sampling elements according to the distribution of the stream.

9.1 Formalism

Definition 9.1 (Stream)**.** A stream σ is a sequence of elements $(s_1, \ldots, s_m)$ where each $s_i \in \{1, \ldots, n\}$. Typically, the length of the stream is unknown and n is large but $n << m$.

Definition 9.2 (Streaming Algorithm)**.** A streaming algorithm consists of 3 parts. An initialization, a processing step of each s_j, and an output step.

[1]Also known as the 0th moment

[2]This is the heavy-hitters problem.

An ideal streaming algorithm uses $o(m)$ memory and can process each element s_j in little time $(O(1), O(\log m), O(\log n)$, etc.). Why? If the stream uses $O(m)$ memory then it is no better than storing all the elements and computing the ideal value. We want to improve on this. The storage of any stream element s_j is $O(\log n)$ so manipulating it should take approximately that much time.

9.2 Uniform Sampling

Exercise 9.3 (Uniform Sampling)**.** Given a stream σ, for each $i \in \{1, \ldots, n\}$, define $f_i = \left|\{j : s_j = i\}\right|$, the number of occurances of value i in the stream. We can define a probability distribution P on $\{1, \ldots, n\}$ by $p_i = f_i/m$.[3] The goal is to output a single element sampled in accordance to the distribution P.

Example 9.4. For example, consider the stream $\sigma = (1, 3, 4, 5, 5, 2, 1, 1, 1, 7)$. In this case $p_1 = 4/10, p_2 = 1/10$, $p_3 = 1/10, p_4 = 1/10, p_5 = 2/10, p_7 = 1/10$. The goal would be to output a random variable in accordance to this distribution.

As we alluded to earlier, the ability to sample an element from the stream uniformly is rather helpful in many more complicated streaming algorithms. Turns out uniformly sampling an element from a stream is a harder problem than people anticipate. Let's consider a naïve algorithm.

Algorithm 9.5 (Naïve Sampling Algorithm)**.**

- Initialize: For each value $i = 1, \ldots, n$, initialize a counter $f_i \leftarrow 0$. Initialize $m \leftarrow 0$.
- Process s_j: If $s_j = i$, set $f_i \leftarrow f_i + 1$. Increment $m \leftarrow m + 1$.
- Output: Calculate $p_i = f_i/m$. Choose an i according to $P = (p_1, \ldots, p_n)$.

Its not too difficult to see what this algorithm is doing and hence its correctness. It is keeping a count of the number of each element. This is certainly an improvement over storing all the elements in the stream which takes $O(m \log n)$ space to store the entire stream. In this case we store n counters which range between 0 and m so require $O(n \log m)$ space in total. As $n << m$, this is better but not the best we can do.

We are going to improve to using only $O(\log n)$ space. This is necessarily optimal as the output is of size $O(\log n)$.

[3]It is a small exercise to verify that $(p_1, \ldots, p_n)$ defines a probability distribution.

Algorithm 9.6 (Uniform Sampling Algorithm)**.**

- Initialize: Set $x \leftarrow$ null.
- Process s_j: With probability $1/j$, set $x \leftarrow s_j$. Otherwise, do nothing.
- Output: x.

Definitely a simpler algorithm! Let's see why it works. We are going to consider a substream $\sigma_j = (s_1, \ldots, s_j)$, the first j elements of the stream σ.

Theorem 9.7. *Let X_j be the random variable output of Algorithm 9.6 on stream $\sigma_j = (s_1, \ldots, s_j)$. Then $\mathbf{Pr}(X_j = i) = f_i^{(j)}/j$ for each $i = 1, \ldots, n$, where $f_i^{(j)}$ is the number of instances of i in the first j elements.*

Proof. We proceed by induction on j. If $j = 1$, then the algorithm necessarily outputs s_1, and therefore appropriately samples from $\sigma_1 = (s_1)$. For the inductive step, assume correctness up to j. An equivalent way to formulate the algorithm given the inductive step is X_{j+1} remains X_j with probability $j/(j+1)$ and switches with probability $1/(j+1)$.

Notice that if $s_{j+1} = i$, then $f_i^{(j)} = f_i^{(j+1)}$ and for $i' \neq i$, $f_{i'}^{(j)} = f_{i'}^{(j+1)}$. Then, we can calculate

$$\begin{aligned}
\mathbf{Pr}(X_{j+1} = i) &= \mathbf{Pr}(X_{j+1} \text{ switches to } i) + \mathbf{Pr}(X_{j+1} \text{ remains } X_j)\mathbf{Pr}(X_j = i) \\
&= \frac{1}{j+1} + \frac{j}{j+1} \cdot \frac{f_i^{(j)}}{j} \\
&= \frac{f_i^{(j)} + 1}{j+1} \\
&= \frac{f_i^{(j+1)}}{j+1}
\end{aligned} \tag{9.1}$$

For $i' \neq i$, then

$$\begin{aligned}
\mathbf{Pr}(X_{j+1} = i') &= \mathbf{Pr}(X_{j+1} \text{ remains } X_j)\mathbf{Pr}(X_j = i') \\
&= \frac{j}{j+1} \cdot \frac{f_{i'}^{(j)}}{j} \\
&= \frac{f_{i'}^{(j)}}{j+1} \\
&= \frac{f_{i'}^{(j+1)}}{j+1}
\end{aligned} \tag{9.2}$$

thereby proving the inductive step. □

As a consequence of the theorem, it is easy to see the correctness of the algorithm.

The incredible advantage of this algorithm is that the output has no m dependence.[4]

[4] Aside from the ability to sample at probability $1/m$, but let's ignore that for now because this is a harder problem for a different day.