Advanced Algorithms 30 October 2018

Lecture 7: Streaming and Sketching Algorithms I

Lecturer: Mohsen Ghaffari *Scribe: Davin Choo*

1 Streaming and sketching algorithms

Thus far, we have been ensuring that our algorithms run fast. What if our system does not have sufficient memory to store all data to post-process it? For example, a router has relatively small amount of memory while tremendous amount of routing data flows through it. In a memory constrained setting, can one compute something meaningful, possible approximately, with limited amount of memory?

More formally, we now look at a slightly different class of algorithms where data elements from $[n] = \{1, \dots, n\}$ arrive in one at a time, in a stream $S = a_1, \dots, a_m$, where $a_i \in [n]$ arrives in the i^{th} time step. At each step, our algorithm performs some computation[1] and discards the item a_i. At the end of the stream[2], the algorithm should give us a value that approximates some value of interest.

One class of interesting problems is computing moments of a given stream S. For items $j \in [n]$, define f_j as the number of times j appears in a stream S. Then, the k^{th} moment of a stream S is defined as $\sum_{j=1}^{n} (f_j)^k$. When $k = 1$, the first moment $\sum_{j=1}^{n} f_j = m$ is simply the number of elements in the stream S. When $k = 0$, by associating $0^0 = 0$, the zeroth moment $\sum_{j=1}^{n} (f_j)^0$ is the number of distinct elements in the stream S. In this lecture, we will discuss methods to approximate the first and zeroth moments of a given stream S.

1.1 Typical tricks

Before we begin, let us describe two typical tricks used to amplify success probabilities of randomized algorithms. Suppose we have a randomized algorithm A that returns an unbiased estimate of a quantity of interest X with probability $p > 0.5$.

Trick 1: Reduce variance Run j independent copies of A on the same instance I, and return the mean $\frac{1}{j} \sum_{i=1}^{j} A(I)$. While $\mathbb{E}(\frac{1}{j} \sum_{i=1}^{j} A(I)) = \mathbb{E}(A(I)) = X$, the variance drops by a factor of j.

Trick 2: Improve success Run k independent copies of A on the same instance I, and return the median. As each copy of A succeeds (independently) with probability $p > 0.5$, the probability that more than half of them fails (and hence the median fails) drops exponential with respect to k.

Let $\epsilon > 0$ and $\delta > 0$ denote the precision factor and error probabilities respectively. The above-mentioned two tricks can be combined with A (See Algorithm 1) to yield a $(1 \pm \epsilon)$-approximation to X that succeeds with probability $> 1 - \delta$.

Algorithm 1 Robust(A, I, ϵ, δ)

```
C ← ∅                                      ▷ Initialize candidate outputs
for k = O(log 1/δ) times do
    sum ← 0
    for j = O(1/ε²) times do
        sum ← sum + A(I)
    end for
    Add sum/j to candidates C              ▷ Include new sample of mean
end for
return Median of C                         ▷ Return median
```

[1] Usually this is constant time so we ignore the runtime.
[2] In general, the length of the stream, m, may not be known.

2 Warm up: Majority element

Definition 1 ("Majority in a stream" problem). *Given a stream $S = \{a_1, \ldots, a_m\}$ of items from $[n] = \{1, \ldots, n\}$, with an element $j \in [n]$ that appears strictly more than $\frac{m}{2}$ times in S, find j.*

Algorithm 2 MajorityStream($S = \{a_1, \ldots, a_m\}$)

```
guess ← 0
count ← 0
for a_i ∈ S do                                ▷ Items arrive in streaming fashion
    if a_i = guess then
        count ← count + 1
    else if count > 1 then
        count ← count − 1
    else
        guess ← a_i
    end if
end for
return guess
```

Example Consider a stream $S = \{1, 3, 3, 7, 5, 3, 2, 3\}$. The table below shows how *guess* and *count* are updated as each element arrives.

Stream elements	1	3	3	7	5	3	2	3
Guess	1	3	3	3	5	3	2	3
Count	1	1	2	1	1	1	1	1

One can verify that MajorityStream uses $\mathcal{O}(\log n + \log m)$ bits to store *guess* and *counter*.

Claim 2. MajorityStream *correctly finds element $j \in [n]$ which appears $> \frac{m}{2}$ times in $S = \{a_1, \ldots, a_m\}$.*

Proof. (Sketch) Match each *other* element in S with a distinct instance of j. Since j appears $> \frac{m}{2}$ times, at least one j is unmatched. As each matching cancels out *count*, only j could be the final *guess*. □

Remark If no element appears $> \frac{m}{2}$ times, then MajorityStream is not guaranteed to return the most frequent element. For example, MajorityStream($S = \{1, 3, 4, 3, 2\}$) returns 2 instead of 3.

3 Estimating the first moment of a stream

A trivial exact solution would be to use $\mathcal{O}(\log m)$ bits to maintain a counter, incrementing for each element observed. For some upper bound M, consider the sequence $(1+\epsilon), (1+\epsilon)^2, (1+\epsilon)^3, \ldots, (1+\epsilon)^{\log_{1+\epsilon} M}$. For any stream length m, there exists $i \in \mathbb{N}$ such that $(1+\epsilon)^i \leq m \leq (1+\epsilon)^{i+1}$. Hence, to obtain a $(1+\epsilon)$-approximation of the first moment, it suffices to track the exponent i to estimate the length of m. For $\epsilon \in \Theta(1)$, this can be done in $\mathcal{O}(\log \log m)$ bits.

Algorithm 3 Morris($S = \{a_1, \ldots, a_m\}$)

```
x ← 0
for a_i ∈ S do                                ▷ Items arrive in streaming fashion
    r ← Random probability from [0, 1]
    if r ≤ 2^(−x) then                        ▷ If not, x is unchanged.
        x ← x + 1
    end if
end for
return 2^x − 1                                ▷ Estimate m by 2^x − 1
```

Morris is due to [Mor78]. The intuition is that we increase the counter (and hence double the estimate) when we observe 2^x new items in expectation. For analysis, let us denote X_m as the value of counter x after exactly m items arrive.

Theorem 3. $\mathbb{E}[2^{X_m} - 1] = m$. *That is,* MORRIS *is an unbiased estimator for the length of the stream.*

Proof. Equivalently, let us prove $\mathbb{E}[2^{X_m}] = m + 1$, by induction on $m \in \mathbb{N} \setminus \{0\}$. On the first element ($m = 1$), x increments with probability 1, so $\mathbb{E}[2^{X_1}] = 2^1 = m + 1$. Suppose it holds for some $m \in \mathbb{N}$, then

$$\begin{aligned}
\mathbb{E}[2^{X_{m+1}}] &= \sum_{j=1}^{m} \mathbb{E}[2^{X_{m+1}} | X_m = j] \Pr[X_m = j] && \text{Condition on previous value of } X_m \\
&= \sum_{j=1}^{m} (2^{j+1} \cdot 2^{-j} + 2^j \cdot (1 - 2^{-j})) \cdot \Pr[X_m = j] && x \text{ increments with probability } 2^{-j} \\
&= \sum_{j=1}^{m} (2^j + 1) \cdot \Pr[X_m = j] && \text{Simplifying} \\
&= \sum_{j=1}^{m} 2^j \cdot \Pr[X_m = j] + \sum_{j=1}^{m} \Pr[X_m = j] && \text{Splitting the sum} \\
&= \mathbb{E}[2^{X_m}] + \sum_{j=1}^{m} \Pr[X_m = j] && \text{Definition of } \mathbb{E}[2^{X_m}] \\
&= \mathbb{E}[2^{X_m}] + 1 && \textstyle\sum_{i=1}^{m} \Pr[X_m = j] = 1 \\
&= (m + 1) + 1 && \text{Induction hypothesis} \\
&= m + 2
\end{aligned}$$

Note that we sum up to m because $x \in [1, m]$ after m items. □

Claim 4. $\mathbb{E}[2^{2X_m}] = \frac{3}{2}m^2 + \frac{3}{2}m + 1$

Proof. Exercise. □

Claim 5. $\mathbb{E}[(2^{X_m} - 1 - m)^2] \leq \frac{m^2}{2}$

Proof. Exercise. Use the Claim 4. □

Theorem 6. *For* $\epsilon > 0$, $\Pr[|(2^{X_m} - 1) - m| > \epsilon m] \leq \frac{1}{2\epsilon^2}$

Proof.

$$\begin{aligned}
\Pr[|(2^{X_m} - 1) - m| > \epsilon m] &= \Pr[((2^{X_m} - 1) - m)^2 > (\epsilon m)^2] && \text{Square both sides} \\
&\leq \frac{\mathbb{E}[((2^{X_m} - 1) - m)^2]}{(\epsilon m)^2} && \text{Markov's inequality} \\
&\leq \frac{m^2/2}{\epsilon^2 m^2} && \text{By Claim 5} \\
&= \frac{1}{2\epsilon^2}
\end{aligned}$$

□

Remark Using the discussion in Section 1.1, we can run MORRIS multiple times to obtain a $(1 \pm \epsilon)$-approximation of the first moment of a stream that succeeds with probability $> 1 - \delta$. For instance, repeating MORRIS $\frac{10}{\epsilon^2}$ times and reporting the mean $\widehat{m}$, $\Pr[|\widehat{m} - m| > \epsilon m] \leq \frac{1}{20}$.

4 Estimating the zeroth moment of a stream

Trivial exact solutions would be to either use $\mathcal{O}(n)$ bits to track if element exists, or use $\mathcal{O}(m \log n)$ bits to remember the whole stream. Suppose there are D distinct items in the whole stream. In this section, we show that one can in fact make do with only $\mathcal{O}(\log n)$ bits to obtain an approximation of D.

4.1 An idealized algorithm

Consider the following algorithm sketch:

1. Take a uniformly random hash function $h : \{1, \ldots, m\} \to [0, 1]$
2. As items $a_i \in S$ arrive, track $z = \min\{h(a_i)\}$
3. In the end, output $\frac{1}{z} - 1$

Since we are randomly hashing elements into the range $[0, 1]$, we expect the minimum hash output to be $\frac{1}{D+1}$[3], so $\mathbb{E}[\frac{1}{z} - 1] = D$. Unfortunately, storing a uniformly random hash function that maps to the interval $[0, 1]$ is infeasible. As storing real numbers is memory intensive, one possible fix is to discretize the interval $[0, 1]$, using $\mathcal{O}(\log n)$ bits per hash output. However, storing this hash function would still require $\mathcal{O}(n \log n)$ space.

[3]See https://en.wikipedia.org/wiki/Order_statistic

4.2 An actual algorithm for estimating the zeroth moment

Instead of a uniformly random hash function that maps to the interval $[0, 1]$, we randomly select a hash from a family of pairwise independent hash functions.

Definition 7 (Family of pairwise independent hash functions)**.** *$\mathcal{H}_{n,m}$ is a family of pairwise independent hash functions if*

- *(Hash definition): $\forall h \in \mathcal{H}_{n,m}$, $h : \{1, \ldots, n\} \to \{1, \ldots, m\}$*
- *(Uniform hashing): $\forall x \in \{1, \ldots, n\}$, $\Pr_{h \in \mathcal{H}_{n,m}}[h(x) = i] = \frac{1}{m}$*
- *(Pairwise independent) $\forall x, y \in \{1, \ldots, n\}$, $x \neq y$, $\Pr_{h \in \mathcal{H}_{n,m}}[h(x) = i \wedge h(y) = j] = \frac{1}{m^2}$*

Remark For this section, we care only about $m = n$, and write $\mathcal{H}_{n,n}$ as $\mathcal{H}_n$.

Claim 8. *Let n be a prime number. Then, $\mathcal{H}_n = \{h_{a,b} : h(x) = ax + b \mod n, \forall a, b \in \mathbb{Z}_n\}$ is a family of pairwise independent hash functions.*

Proof. (Sketch) For any given a, b,

- There is a unique value of $h(x) \mod n$, out of n possibilities.
- The system $\{ax + b = i \mod n, ay + b = j \mod n\}$ has a unique solution for (x, y), out of n^2 possibilities.

□

Remark If n is not a prime, we know there exists a prime p such that $n \leq p \leq 2n$, so we round n up to p. Storing a random hash from $\mathcal{H}_n$ is then storing the numbers a and b in $\mathcal{O}(\log n)$ bits.

We now present an algorithm [FM85] which estimates the zeroth moment of a stream and defer the analysis to the next lecture. In FM, ZEROS refer to the number of trailing zeroes in the binary representation of $h(a_i)$. For example, if $h(a_i) = 20 = (...10100)_2$, then $\text{ZEROS}(h(a_i)) = 2$.

Algorithm 4 FM($S = \{a_1, \ldots, a_m\}$)

```
h ← Random hash from H_{n,n}
Z ← 0
for a_i ∈ S do                                  ▷ Items arrive in streaming fashion
    Z = max{Z, ZEROS(h(a_i))}    ▷ ZEROS(h(a_i)) = # trailing zeroes in binary representation of h(a_i)
end for
return 2^Z · √2                                 ▷ Estimate of D
```

References

[FM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.