PRINCETON UNIV. F'23 COS 521: ADVANCED ALGORITHM DESIGN

Lecture 9: Approximate Counting

Lecturer: *Huacheng Yu*

In streaming algorithms, we study how to "process large data using small space." In this lecture, the most important parameter of an algorithm is its memory usage. Streaming algorithms are applicable to many situations where the input data is presented to the algorithm sequentially, while the algorithm does not have enough space to store the whole input. Therefore, we must process the data as they arrive, while maintaining a small-sized data structure in memory.

The input to a streaming algorithm is a stream $(a_1, \ldots, a_n)$ of items, and the algorithm can only access the stream in order. There are many different problems that can be solved in this model using small space. Today, we will study one of the most basic question in stream – count the number of items in the stream approximately.

More formally, we wish to maintain a counter n, initialized to 0, supporting two operations:

- `inc()`: set $n \leftarrow n + 1$ (promised that n never exceeds N)
- `query()`: return an approximation $\tilde{n} = (1 \pm \varepsilon)n$ with probability at least $1 - \delta$

The trivial algorithm, which maintains n exactly, uses $O(\log N)$ bits of space. Here, we are interested in understanding if insisting on using much smaller memory, whether we can still obtain a very close approximation with high probability.

1 The "offline" question

Let us first consider an easier version of the problem: Given an integer $n \in [N]$, encode n using $\ll \log N$ bits such that from the encoding one can recover an $\tilde{n} \in [n/2, 2n]$. This is an easier question, because if we have an approximate counter using S bits of space, then by doing n `inc()` operations, we obtain a memory state of S bits, from which we can recover an $\tilde{n} \approx n$. Thus, this memory state can be viewed as a succinct encoding of n.

For this easier question, the idea is to use one encoding to represent an interval of values. One such strategy is to note that if we round n to a power of two, then this is a 2-approximation. Suppose $2^X \le n < 2^{X+1}$, then by using the value of X as the encoding, and returning $\tilde{n} = 2^X$, we obtain the desired succinct encoding that uses only $O(\log \log N)$ bits.

2 The update algorithm

We will maintain such an X in memory, and the remaining question is how to update X when we increment n. First observe that X represents that the counter value is approximately $\tilde{n} = 2^X$. As we increase n, we need to increment X at some point. When we increment X to $X + 1$, the counter value it reprsents becomes 2^{X+1}, i.e., suddenly it is increased by 2^X.

The idea of the update algorithm is to increase X with probability 2^{-X} to offset this difference. For simplicity of notations when $n = 0$, in the actual algorithm, we will in fact think X represents $\tilde{n} = 2^X - 1$. The full algorithm is shown below.

Init()

1. set $X \leftarrow 0$

Inc()

1. set $X \leftarrow X + 1$ with probability 2^{-X}; do nothing otherwise

Query()

1. return $\tilde{n} = 2^X - 1$

The analysis of the algorithm has two steps: we first prove that the output has the correct expectation, then show that its variance is bounded. Let X_n be the random variable denoting the value of X after n `inc()`.

Claim 1. *We have* $\mathbb{E}[2^{X_n} - 1] = n$.

Proof. We first have $X_0 = 0$, i.e., $\mathbb{E}[2^{X_0} - 1] = 0$.

Then for general $n > 0$, we have

$$\begin{aligned}\mathbb{E}[2^{X_n} \mid X_{n-1}] &= 2^{-X_{n-1}} \cdot 2^{X_{n-1}+1} + (1 - 2^{-X_{n-1}}) \cdot 2^{X_{n-1}} \\ &= 2 + 2^{X_{n-1}} - 1 \qquad\qquad = 2^{X_{n-1}} + 1.\end{aligned}$$

Taking the expectation over X_{n-1} on both sides, we get

$$\mathbb{E}[2^{X_n}] = \mathbb{E}[2^{X_{n-1}}] + 1.$$

The claim holds by induction. □

Next, we bound its variance.

Claim 2. *We have* $\mathrm{Var}[2^{X_n} - 1] \le O(n^2)$.

Proof. We have $\mathrm{Var}[2^{X_n} - 1] = \mathrm{Var}[2^{X_n}] = \mathbb{E}[4^{X_n}] - (\mathbb{E}[2^{X_n}])^2$. The previous claim proved that $\mathbb{E}[2^{X_n}] = n + 1$. Now we calculate $\mathbb{E}[4^{X_n}]$. First we have $4^{X_0} = 1$. For $n > 0$, we have

$$\begin{aligned}\mathbb{E}[4^{X_n} \mid X_{n-1}] &= 2^{-X_{n-1}} \cdot 4^{X_{n-1}+1} + (1 - 2^{-X_{n-1}}) \cdot 4^{X_{n-1}} \\ &= 4 \times 2^{X_{n-1}} + 4^{X_{n-1}} - 2^{X_{n-1}} \\ &= 4^{X_{n-1}} + 3 \times 2^{X_{n-1}}.\end{aligned}$$

Taking expectation on both sides, we get

$$\mathbb{E}[4^{X_n}] = \mathbb{E}[4^{X_{n-1}}] + 3\mathbb{E}[2^{X_{n-1}}] = \mathbb{E}[4^{X_{n-1}}] + 3n.$$

Thus, $\mathbb{E}[4^{X_n}] = \frac{3}{2}n^2 + \frac{3}{2}n + 1$, and $\mathrm{Var}[2^{X_n} - 1] \le O(n^2)$. □

By Chebyshev's inequality, we obtain that

$$\Pr[|\tilde{n} - n| \ge T] \le O\left(\frac{n^2}{T^2}\right).$$

That is, for $T = C \cdot n$ for large constant C, $\tilde{n}$ is at most $O(n)$ with probability 0.9.

3 Reducing the variance

The output of the above algorithm has the right expectation, but its variance is too large to give an $(1 \pm \varepsilon)$-approximation. To reduce the variance, we can take multiple independent copies and output their average.

Formally, we maintain s independent copies of the algorithm (for some parameter s): each time we need to increment n, we independently run `inc()` for each copy; when it is queried, we query each copy and output $\tilde{n}$ equal to the average value of $2^X - 1$. Let $X^{(i)}$ be the value of X in the i-th copy. Then we have

$$\mathbb{E}\left[\frac{1}{s}\sum_{i=1}^{s}\left(2^{X^{(i)}} - 1\right)\right] = n.$$

Its variance is

$$\begin{aligned}\mathrm{Var}\left[\frac{1}{s}\sum_{i=1}^{s}\left(2^{X^{(i)}} - 1\right)\right] &= \frac{1}{s^2}\mathrm{Var}\left[\sum_{i=1}^{s}\left(2^{X^{(i)}} - 1\right)\right] \\ &= \frac{1}{s} \cdot \mathrm{Var}[2^{X^{(1)}} - 1] \\ &= O(n^2/s).\end{aligned}$$

Therefore, Chebyshev's inequality gives

$$\Pr[|\tilde{n} - n| \geq T] \leq O\left(\frac{n^2}{sT^2}\right).$$

By setting $s = O(\varepsilon^{-2} \cdot \delta^{-1})$, for $T = \varepsilon n$, we have

$$\Pr[\tilde{n} = (1 \pm \varepsilon)n] > 1 - \delta.$$

That is, the space usage of the algorithm is $O(s \cdot \log\log N) = O(\varepsilon^{-2} \cdot \delta^{-1} \cdot \log\log N)$. This is $O(\log\log N)$ for constant ε and δ.

4 Median of Means

One common strategy to improve the dependence on the failure probability δ is to use "median-of-means". That is, we maintain $s_1 \cdot s_2$ copies of the algorithm, which are divided into s_1 groups of size s_2. Denote by $X^{(i,j)}$ the value of X in j-th copy in group i, and $\tilde{n}_{i,j} = 2^{X^{(i,j)}} - 1$ be its output. When it is queried, we first compute the average of each group: $\tilde{n}_i = \frac{1}{s_2}\sum_{j=1}^{s_2} \tilde{n}_{i,j}$. Then we output the median of the averages: $\tilde{n} = \mathrm{median}(\tilde{n}_1, \ldots, \tilde{n}_{s_1})$.

When s_2 is set to $C \cdot \varepsilon^{-2}$ for a sufficiently large constant C, the argument from the previous section gives that

$$\Pr[\tilde{n}_i = (1 \pm \varepsilon)n] > 3/4,$$

for each i. To see what is the probability that their median is also a $(1 \pm \varepsilon)$-approximation, observe that

- the median is larger than $(1+\varepsilon)n$, if and only if
- there are at least $s_1/2$ averages $\tilde{n}_i$ that are larger than $(1+\varepsilon)n$.

However, for each $\tilde{n}_i$, this happens with probability at most $1/4$. This allows us to apply Chernoff bound: Let Y_i indicate if $\tilde{n}_i > (1+\varepsilon)n$, then $\Pr[Y_i = 1] < 1/4$, and all Y_i are independent. Chernoff bound implies that $\Pr[\sum_i Y_i \geq s_1/2] < \exp(-\Theta(s_1))$, i.e., the median is larger than $(1+\varepsilon)n$ with probability at most $\exp(-\Theta(s_1))$. Similarly, we can obtain the same bound on the probability that the median is smaller than $(1-\varepsilon)n$.

By setting $s_1 = C\log(1/\delta)$ for a sufficiently large constant C, this implies that

$$\Pr[\tilde{n} = (1 \pm \varepsilon)n] > 1 - \delta.$$

Now the total number of copies we maintain is only $O(\varepsilon^{-2}\log(1/\delta))$. That is, we use space $O(\varepsilon^{-2}\log(1/\delta)\log\log N)$ bits.

5 Morris Counter

The Morris counter is similar to the algorithm we saw above. It has a parameter $\alpha \in (0,1)$ such that X is set to $X+1$ with probability $(1+\alpha)^{-X}$, and outputs $\tilde{n} = \left((1+\alpha)^X - 1\right)/\alpha$. We will prove in the problem set that a single Morris counter achieves space $O(\log(1/\varepsilon) + \log\log N + \log\log(1/\delta))$ by choosing the right parameter α.