

1 Streaming Algorithms

Randomness is useful for sampling! This is what statisticians do, i.e. for political polls. Law of Large Numbers and Chernoff bounds demonstrate that samples converge quickly to the population (ignoring the issue of how to actually choose a random sample).

Streaming algorithms are essentially a pure use of sampling, with a few added twists. In the Streaming Algorithm Model, we have a stream of data coming at us that we wish to process in some way. There is more data than can feasibly be stored, but we can read the data and would like to compute some statistics on it. In this model, we expect that we cannot spend very much time per unit of data. A motivating example is a web server, in which there is a stream of clicks or URL requests that may be processed.

2 Heavy Hitters

In the heavy hitters problem, we wish to find all values x such that x occurs “approximately” m or more times. A motivating example would be if we wished to find a list of regular customers of a website so that we could cache their profiles for faster access, or otherwise to detect bots or spam.

More specifically, on input list $x_1, x_2, \dots, x_n$ we want a list of elements such that with high probability, we have the following:

- If x is in the list, $x = x_i$ for at least $\frac{m}{2}$ x_i ’s
- If $x = x_i$ for at least m x_i ’s, then x is in the list.

Suppose that each x_i has length at most ℓ , i.e. $|x_i| \leq \ell$. We would like to minimize the processing time per element x_i , i.e. it should probably be constant time. For space usage, a necessary lower bound is $\Omega\left(\frac{n}{m} \cdot \ell\right)$, but we’ll actually achieve something more like $O\left(\frac{n}{m} \cdot \ell \cdot \log \frac{n}{m}\right)$. We are most interested in the case when $m = \alpha \cdot n$, i.e. when m is a constant fraction of n . Note that the case when m is constant is degenerate.

A related problem is Distinct Elements which is sort of a dual of Heavy Hitters in which we wish to count the number of distinct elements among the x_i . This is easy if we can store a hash table, but what if not...?

2.1 Sampling Algorithm for Heavy Hitters

- Pick each element with probability $\frac{L}{m}$, where we'll eventually pick $L \approx \log \frac{n}{m}$.
- If an element is chosen, we enter it in the hash table and keep a counter.
- If the element isn't chosen, but is in the hash table, increment the associated counter (actually not necessary to "flip a coin" for such elements, but may be easier to conceptualize if we consider that we do).
- At the end, output those elements with counters at least $m/2$.

2.2 Analysis of Sampling Algorithm for Heavy Hitters

Since only selected elements are added to the hash table (and repeats are only added once), $E(\text{Size of Hash Table}) \leq \frac{n \cdot L}{m}$. Using Chernoff bounds, it is easy to see that the size of the table is less than $\frac{2 \cdot n \cdot L}{m}$ with high probability. Thus it is sufficient to reserve $\frac{2 \cdot n \cdot L}{m}$ spots in the hash table. Hashing will be constant time for both insertion and increment.

The running time is thus linear, i.e. $O(n)$, which is the best that we could expect, and the space is inflated only by a factor of $2L$, i.e. space is $O\left(\frac{n}{m} \cdot \ell \cdot L\right)$.

Regarding correctness, the first statement is all true. That is, we only return x if x occurs at least $\frac{m}{2}$ times. To analyze the second statement, consider the probability that x occurs at least m times and we fail to return it.

$$\Pr \left[x \text{ not picked first } \frac{m}{2} \text{ times} \right] \leq \left(1 - \frac{L}{m} \right)^{m/2} \leq e^{-L/2} \leq e^{-k} \cdot \frac{m}{n}$$

Where the last inequality holds when we choose $L > \ln \frac{n}{m} + 2k$. Note that if we chose $k = 0$, this bound is just 1, but for larger k , the bound improves. Thus if we want a success probability of $1 - \delta$, the space required is $O\left(\frac{n}{m} \cdot \ell \cdot \left(\log \frac{n}{m} + \log \frac{1}{\delta}\right)\right)$

2.3 Variant of Heavy Hitters in which n is unknown

An interesting variant of the problem is when we are not given n in advance, and rather than getting m , we are given α such that $m = \alpha \cdot n$. In this case, the adapted algorithm is to choose the t -th element with probability $\frac{L}{\alpha \cdot t}$, then we get a loss of $\frac{1}{\alpha} \sum_{i=1}^n \frac{1}{t} = O\left(\frac{1}{\alpha} \log n\right)$ in table size for not knowing n in advance. Correctness still holds. Thus for unknown n , and $m = \alpha \cdot n$, the space usage is $O\left(\frac{1}{\alpha} \cdot \ell \cdot \log n \cdot \left(\log \frac{1}{\alpha} + \log \frac{1}{\delta}\right)\right)$.

3 Distinct Elements

Recall that in the Distinct Elements problem, on streaming input $x_1, x_2, \dots, x_n$ we wish to compute $|\{x_i \mid i \in \{1, \dots, n\}\}|$.

Our goal is to construct a randomized algorithm that maintains a subset S such that each element in $\{x_i \mid i \in \{1, \dots, n\}\}$ is added to S with probability p , but we must first acknowledge a number of complications.

- We need to keep the size of S very small, so we must set p conservative. We may dynamically adjust p as the need arises.
- We need to decide how to pick elements with probability p . One method is to construct a randomly chosen map h from x 's to coin flips, i.e. $h : X \rightarrow \text{coin flips}$. Thus once we pick randomness to construct h , coin flips for x_i depend only on the value of x_i , i.e. if $x_i = x_j$ then identical coin flips will be used, but if $x_i \neq x_j$, the coin flips will be uncorrelated.

To implement such an h , we need either a pairwise independent or universal hash function. For our purposes, pairwise independence should be sufficient. Say that x is an ℓ -bit string, and r is an ℓ -bit random string.

$$x = \begin{pmatrix} y_1 \\ \vdots \\ y_\ell \end{pmatrix}, r = \begin{pmatrix} r_1 \\ \vdots \\ r_\ell \end{pmatrix}$$

Then to get one random bit we could construct h as follows.

$$h_r(x) = \sum_{i=1}^{\ell} y_i \cdot r_i \pmod{2}$$

But if we want ℓ bits of randomness, we must choose ℓ such vectors, i.e. let H be an $\ell \times \ell$ matrix of random bits, and define h as follows.

$$h_H(x) = x \cdot H$$

If we want $p = \frac{1}{2}$, then we would choose x if the first bit of $h(x)$ is 0; if we want $p = \frac{1}{4}$, then we would choose x if the first 2 bits of $h(x)$ are 0; and if we want $p = \frac{1}{2^k}$, then we would choose x if the first k bits of $h(x)$ are 0.

We will keep track of the largest number of initial zeros in any $h(x)$. Say that this number is J , then we estimate that $|\{x_i \mid i \in \{1, \dots, n\}\}| \approx 2^J$. With reasonable probability, this is pretty good, but reasonable is not great, so actual algorithm constructs many hash functions h with different tables, and the median of the results is returned.

Claim 1. *If $|\{x_i \mid i \in \{1, \dots, n\}\}| > c \cdot 2^{J'}$, then $\Pr[J \geq J'] > 1 - 1/c$, and if $|\{x_i \mid i \in \{1, \dots, n\}\}| < 2^{J'}/c$, then $\Pr[J \leq J'] > 1 - 1/c$.*

Proof. Say that $|\{x_i \mid i \in \{1, \dots, n\}\}| < 2^{J'}/c$, then for each element x , the probability that $h(x)$ has J' initial zeros is exactly $1/2^{J'}$. Thus we have the following (where the $\leq$ is by a union bound, which holds independent of correlation).

$$\begin{aligned} \Pr[J \geq J'] &= \Pr[\exists x \in \{x_1, \dots, x_n\} \text{ s.t. } h(x) \text{ has } J' \text{ initial 0's}] \\ &\leq |\{x_i \mid i \in \{1, \dots, n\}\}| \cdot 1/2^{J'} \leq 1/c \end{aligned}$$

Say that $\{x_1, \dots, x_n\} = \{y_1, \dots, y_j\}$ for y_i 's distinct, i.e. say that the indexes are into a sorted lists of the x_i 's. Let E_j be the event that $h(y_j)$ has J initial 0's, then $\Pr[E_j] = (1/2)^J$, and E_j and E_k are uncorrelated for $j \neq k$. Let $E = \sum_j E_j$, then we have the following.

$$\begin{aligned} \text{Var}[E] &= \text{Exp}[(E - \text{Exp}[E])^2] = \text{Exp}[E^2] - \text{Exp}[E]^2 \\ \text{Exp}[E] &= c \cdot 2^{J'}/2^{J'} = c \\ \text{Exp}[E^2] &= \text{Exp}\left[\left(\sum_j E_j\right)^2\right] = \text{Exp}\left[\sum_j E_j^2 + 2 \cdot \sum_{j \neq k} E_j \cdot E_k\right] = c + \end{aligned}$$

□