
Lectures 13, 14

1 Streaming Algorithms

A data stream is a very large sequence of inputs (x_1, x_2, \dots, x_m) drawn from a universe of size n , realtime data like server logs, user clicks and search queries are modeled by streams. The stream is too large to be stored in memory and must be processed in a single pass. A streaming algorithm processes a data stream in a single pass using space $O(\log^c n)$.

1.1 Counting distinct elements

Consider the problem of finding the number of unique visitors to a website, by processing a data stream consisting of the ip addresses of visitors. The problem is equivalent to estimating the number of distinct items in the stream.

Exactly counting the number of distinct elements requires $O(n)$ space, we will present a randomized algorithm for estimating the distinct elements to up to a multiplicative factor of $(1 \pm \epsilon)$ using $\text{poly}(\log n, \frac{1}{\epsilon})$ space.

1.1.1 Exact counting requires $O(n)$ space

Suppose A is an algorithm that counts the number of distinct elements in a stream S with elements drawn from $[n]$. After executing A on the input stream S it acts as a membership tester for S . On input $x \in [n]$ the count of distinct items increases by 1 if $x \notin S$ and stays the same if $x \in S$. The internal state of A must contain enough information to distinguish between the 2^n possible subsets of $[n]$ that could have occurred in S . The algorithm requires $O(n)$ bits of storage to distinguish between 2^n possibilities.

1.1.2 Approximate counting

The approximate counting algorithm is probabilistic, it uses internal random bits to perform well with high probability against any stream S . The probabilities are over the internal randomness used by the algorithm, the input stream is deterministic and fixed in advance.

A toy problem: Output ‘yes’ if the number of distinct items N is greater than $2k$, ‘no’ if N is less than k , we do not care about the output if $k \leq N \leq 2k$.

Solution: Choose a uniformly random hash function $h : [n] \rightarrow [B]$, where the number of buckets $B = O(k)$. Output ‘yes’ if there is some $x_i \in S$ such that $h(x_i) = 0$ else output ‘no’.

The value $h(x)$ is uniformly distributed on $[B]$, so for all $x \in U$ we have $\Pr_{h \in \mathcal{H}}[h(x) = 0] = 1/B$. If the number of items in the stream is at most k , the probability that none of the N items hash to 0 is,

$$\Pr[A(x) = No \mid N \leq k] = \left(1 - \frac{1}{B}\right)^N \geq \left(1 - \frac{1}{B}\right)^k$$

If the number of elements is greater than $2k$ then the probability that the algorithm outputs no is,

$$\Pr[A(x) = \text{No} \mid N > 2k] = \left(1 - \frac{1}{B}\right)^N \leq \left(1 - \frac{1}{B}\right)^{2k}$$

The gap between the probability of the output being ‘no’ for the two cases is a constant for $B = O(k)$.

However, specifying a random hash function requires $O(n \log B)$ bits of storage, the truth table must be stored to evaluate the hash function. The memory requirement can be reduced by choosing h from a hash function family \mathcal{H} of small size having good independence properties.

2-wise independent hash functions: The property required from \mathcal{H} is 2-wise independence, informally a hash function family is 2 wise independent if the hash value $h(x)$ provides no information about $h(y)$.

CLAIM 1

The family $\mathcal{H} : [n] \rightarrow [B]$ of hash functions $h_{a,b}(x) = (ax + b \bmod p) \bmod B$ where p is a prime number larger than n and $a, b \in \mathbb{Z}_p$ is 2-wise independent,

$$\Pr_{a,b}[h(x) = c \wedge h(y) = d] \approx \frac{1}{B^2} \quad \forall x \neq y$$

PROOF: If $h(x) = c$ and $h(y) = d$ then there exist $i, j \in \mathbb{Z}$ such that $c + iB, d + jB \in \mathbb{Z}_p$ and the following linear equations over \mathbb{Z}_p are satisfied,

$$ax + b = c + iB \quad ay + b = d + jB$$

The linear system has a unique solution as the determinant $(x - y) \neq 0$ for distinct x, y . The number of solutions is equal to the number of pairs i, j such that $c + iB, d + jB \in \mathbb{Z}_p$, which is independent of c, d (apart from small error terms). \square

We analyze the algorithm using a random hash function from \mathcal{H} with $B = 4k$ buckets. Arguing similarly to claim 1, it follows that $\Pr_{a,b}[h(x) = 0] = 1/B$ for all $x \in [U]$. If there are k elements in the stream the probability of some element being hashed to 0 can be bounded using the union bound $\Pr[\cup A_i] \leq \sum \Pr[A_i]$,

$$\Pr[A(x) = \text{Yes} \mid N < k] \leq \frac{k}{B} = \frac{1}{4} \tag{1}$$

The inclusion exclusion principle is used to show that the probability of the output being yes is large if there are more than $2k$ elements in the stream. Truncating the inclusion exclusion formula to the first two terms yields $\Pr[\cup A_i] \geq \sum \Pr[A_i] - \sum \Pr[A_i \cap A_j]$,

$$\Pr[A(x) = \text{Yes} \mid N \geq 2k] \geq \frac{2k}{B} - \frac{2k \cdot (2k - 1)}{2B} \geq \frac{2k}{B} \left(1 - \frac{k}{B}\right) = \frac{3}{8} \tag{2}$$

The yes and no cases are separated by a gap of $1/8$, the memory used by the algorithm is $O(\log n)$ as numbers a, b need to be stored. Using a combination of standard tricks, the quality of approximation can be improved to $1 \pm \epsilon$.

1.2 A $1 \pm \epsilon$ approximation:

The probability of obtaining a correct answer is boosted to $1 - \delta$ by running the algorithm with several independent hash functions using the following simplified version of Chernoff bounds,

CLAIM 2

If a coin with bias b is flipped $k = O(\frac{\log(1/\delta)}{\epsilon^2})$ times, with probability $1 - \delta$ the number of heads \hat{b} satisfies $bk(1 - \epsilon) \leq \hat{b} \leq bk(1 + \epsilon)$.

The algorithm is run for $O(\log 1/\delta)$ independent iterations and the output is ‘yes’ if the fraction of yes answers is more than $5/16$. Applying the claim for the yes and no cases, it follows that the correct answer is obtained with probability at least $1 - \delta$.

The number of distinct items N can be approximated to a factor of 2 using the binary search trick. The algorithm is run simultaneously for the $\log n$ intervals $[2^k, 2^{k+1}]$ for $k \in [\log n]$. If $N \in [2^k, 2^{k+1}]$ then with high probability the first $k - 1$ runs answer ‘yes’, the answer for the k -th run is indeterminate and the last $\log n - k - 1$ runs answer ‘no’. The first no in the sequence of answers occurs either for $[2^k, 2^{k+1}]$ or $[2^{k+1}, 2^{k+2}]$, the left end point of the interval where the transition occurs satisfies $\frac{N}{2} \leq L \leq 2N$.

The third trick is to replace 2 by $1 + \epsilon$ in equations (??), (??) and change parameters appropriately in the boosting part to approximate the number of distinct items in the stream up to a factor of $1 \pm \epsilon$.

The space requirement of the algorithm is $O(\log n \cdot \log_{1+\epsilon} n \cdot \frac{\log(1/\delta)}{\epsilon^2})$, the $\log n$ is the amount of memory required to store a single hash function, the $\log_{1+\epsilon} n$ is the number of intervals considered and $\frac{\log(1/\delta)}{\epsilon^2}$ is the number of independent hash functions used for each interval.

2 Sketching

We solved the problem of approximating the number of distinct elements in the stream. More generally we would like to compute statistics like the mean and variance in the streaming model.

Let i_j be the j -th item and m_j be the number of occurrences of the i_j in the stream. The first moment $\sum_i m_i$ is trivial to compute, consider the problem of estimating $\mu := \sum_i m_i^2$ in the streaming model.

The hash function h is chosen from a 4-wise independent family $\mathcal{H} : [n] \rightarrow \pm 1$. The algorithm outputs $Z = (\sum_i h(x_i))^2$ as an estimate for μ , the memory requirement is $O(\log n)$. The analysis will show that $E[Z^2] = \mu$ and that the variance is small. Denoting the hash value for the j -th item $h(i_j)$ by Y_j we have,

$$Z = \sum_{i \in [n]} h(x_i) = \sum_{j \in N} Y_j m_j$$

The expectation of Z^2 can be computed by squaring and using the 2 wise independence of the hash function to cancel out the cross terms,

$$E[Z^2] = \sum_j E[Y_j^2] m_j^2 + \sum_{i,j} E[Y_i] E[Y_j] m_i m_j = \sum_i m_i^2 = \mu$$

A variance calculation is required to ensure that we obtain the correct answer with sufficiently high probability. Recall that the variance of a random variable X is equal to $E[X^2] - E[X]^2$, the variance calculation requires computing the fourth moment of Z ,

$$E[Z^4] = \sum_i E[Y_i^4 m_i^4] + 6 \sum_{i,j} E[Y_i^2 Y_j^2 m_i^2 m_j^2] = \sum_i m_i^4 + 6 \sum_{i,j} m_i^2 m_j^2$$

The variance calculation shows that $Var(Z^2) = 4 \sum m_i^2 m_j^2 \leq 2\mu^2$. The Chebyshev inequality is useful for bounding the deviation of a random variable from its mean,

$$\Pr[|X - \mu| \geq \epsilon\mu] \leq \frac{Var(X)}{\epsilon^2 \mu^2}$$

The variance is too large for Chebyshev's inequality to be useful. The variance can be reduced by running the procedure over $k = 2/\delta\epsilon^2$ independent iterations, with the output being $Z = \frac{1}{k} \sum_{i \in [k]} Z_i^2$.

The expectation $E[Z] = \mu$ by linearity and the variance can be calculated using $Var(X + Y) = Var(X) + Var(Y)$ for independent random variables X and Y .

$$Var[Z] = \sum_{i \in [k]} Var \left[\frac{Z_i^2}{k} \right] \leq \frac{2\mu^2}{k}$$

Applying the Chebyshev inequality for $Z = \frac{1}{k} \sum_{i \in [k]} Z_i^2$ with $k = \frac{2}{\delta\epsilon^2}$ yields $\Pr[|Z - \mu| \geq \epsilon\mu] \leq \delta$. The output of the algorithm Z is therefore a $(1 \pm \epsilon)$ approximation for μ with probability at least $1 - \delta$. The memory requirement for the algorithm is $O(\log n/\epsilon^2)$.