54 Streaming Algorithms

For this chapter, we will consider the streaming model. In this setting, the input is presented as a ''stream'' which we can read *in order*. In particular, at each step, we can do some processing, and then move forward one unit in the stream to read the next piece of data. We can choose to read the input again after completing a ''pass'' over it.

There are two measures for the performance of algorithms in this setting. The first is the number of passes we make over the input, and the second is the amount of memory that we consume. Some interesting special cases are:

- 1 pass, and $O(1)$ memory: This is equivalent to computing with a DFA, and hence we can recognise only regular languages.
- 1 pass, and unbounded memory: We can store the entire stream, and hence this is just the traditional computing model.

54.1 Frequent Elements

For this problem, the input is a stream $\alpha[1 \dots m]$ where each $\alpha[i] \in [n]$. We define for each $j \in [n]$ the *frequency* f_j which counts the occurences of j in $\alpha[1 \dots m]$. Then the majority problem is to find (if it exists) a j such that $f_j > m/2$.

We consider the more general frequent elements problem, where we want to find $F_k = \{j \mid f_j > m/k\}$. Suppose that we knew some small set C which contains F_k. Then, with a pass over the input, we can count the occurrences of each element of C, and hence find F_k in $\mathcal{O}(|C| \log m)$ space.

The Misra/Gries Algorithm

We will now see a deterministic one-pass algorithm that estimates the frequency of each element in a stream of integers. We shall see that it also provides us with a small set C containing F_k, and hence lets us solve the frequent elements problem efficiently.

Algorithm FrequencyEstimate (Misra/Gries Algorithm)
Input: the data stream α, the target for the estimator k.

1. *Init*: $A \leftarrow \emptyset$. ◁ *an empty map*
2. *Process*(x):
3. If $x \in \text{keys}(A)$, $A[x] \leftarrow A[x] + 1$.
4. Else If $|\text{keys}(A)| < k - 1$, $A[x] \leftarrow 1$.
5. Else $\forall a \in \text{keys}(A)$: $A[a] \leftarrow A[a] - 1$, delete a from A if $A[a] = 0$.

Output: $\hat{f}_a = A[a]$ If $a \in \text{keys}(A)$, and $\hat{f}_a = 0$ otherwise.

Let us show that $\hat{f}_a$ is a good estimate for the frequency f_a.

Lemma: $f_a - m/k \leq \hat{f}_a \leq f_a$

Proof: We see immediately that $\hat{f}_a \leq f_a$, since it is only incremented when we see a in the stream.

To see the other inequality, suppose that we have a counter for each $a \in [n]$ (instead of just $k-1$ keys at a time). Whenever we have at least k non-zero counters, we will decrease all of them by 1; this gives exactly the same estimate as the algorithm above.

Now consider the potential function $\Phi = \sum_{a \in [n]} A[a]$. Note that Φ increases by exactly m (since α contains m elements), and is decreased by k every time any $A[x]$ decreases. Since $\Phi = 0$ initially and $\Phi \geq 0$, we get that $A[x]$ decreases at most m/k times. □

Theorem: There exists a deterministic 2-pass algorithm that finds F_k in $\mathcal{O}(k(\log n + \log m))$ space.

Proof: In the first pass, we obtain the frequency estimate $\hat{f}$ by the Misra/Gries algorithm. We set $C = \{a \mid \hat{f}_a > 0\}$. For $a \in F_k$, we have $f_a > m/k$, and hence $\hat{f}_a > 0$ by the previous Lemma. In the second pass, we count f_c exactly for each $c \in C$, and hence know F_k at the end.

To see the bound on space used, note that $|C| = |\text{keys}(A)| \leq k-1$, and a key-value pair can be stored in $\mathcal{O}(\log n + \log m)$ bits. □

The Count-Min Sketch

We will now look at a randomized streaming algorithm that solves the frequency estimation problem. While this algorithm can fail with some probability, it has the advantage that the output on two different streams can be easily combined.

Algorithm FrequencyEstimate (Count-Min Sketch)
Input: the data stream α, the accuracy ε, the error parameter δ.

1. *Init*: $C[1 \ldots t][1 \ldots k] \leftarrow 0$, where $k \leftarrow \lceil 2/\varepsilon \rceil$ and $t \leftarrow \lceil \log(1/\delta) \rceil$.
2. Choose t independent hash functions $h_1, \ldots, h_t : [n] \to [k]$, each from a 2-independent family.
3. *Process*(x):
4. For $i \in [t]$: $C[i][h_i(x)] \leftarrow C[i][h_i(x)] + 1$.

Output: Report $\hat{f}_a = \min_{i \in t} C[i][h_i(a)]$.

Note that the algorithm needs $\mathcal{O}(tk \log m)$ bits to store the table C, and $\mathcal{O}(t \log n)$ bits to store the hash functions $h_1, \ldots, h_t$, and hence uses $\mathcal{O}(1/\varepsilon \cdot \log(1/\delta) \cdot \log m + \log(1/\delta) \cdot \log n)$ bits. It remains to show that it computes a good estimate.

Lemma: $f_a \leq \hat{f}_a \leq f_a + \varepsilon m$ with probability δ.

Proof: Clearly $\hat{f}_a \geq f_a$ for all $a \in [n]$; we will show that $\hat{f}_a \leq f_a + \varepsilon m$ with probability at least δ. For a fixed element a, define the random variable

$$X_i := C[i][h_i(a)] - f_a$$

For $j \in [n] \setminus \{a\}$, define the indicator variable $Y_{i,j} := [h_i(j) = h_i(a)]$. Then we can see that

$$X_i = \sum_{j \neq a} f_j \cdot Y_{i,j}$$

Note that $\mathbb{E}[Y_{i,j}] = 1/k$ since each h_i is from a 2-independent family, and hence by linearity of expectation:

$$\mathbb{E}[X_i] = \frac{||f||_1 - f_a}{k} = \frac{||f_{-a}||_1}{k}$$

And by applying Markov's inequality we obtain a bound on the error of a single counter:

$$\Pr[X_i > \varepsilon \cdot m] \geq \Pr[X_i > \varepsilon \cdot ||f_{-a}||_1] \leq \frac{1}{k\varepsilon} \leq 1/2$$

Finally, since we have t independent counters, the probability that they are all wrong is:

$$\Pr\left[\bigcap_i X_i > \varepsilon \cdot m\right] \leq 1/2^t \leq \delta$$

□

The main advantage of this algorithm is that its output on two different streams (computed with the same set of hash functions h_i) is just the sum of the respective tables C. It can also be extended to support events which remove an occurence of an element x (with the caveat that upon termination the "frequency" f_x for each x must be non-negative). (TODO: perhaps make the second part an exercise?).

54.2 Counting Distinct Elements

We continue working with a stream $\alpha[1 \dots m]$ of integers from $[n]$, and define f_a (the frequency of a) as before. Let $d = |\{j : f_j > 0\}|$. Then the distinct elements problem is to estimate d.

The AMS Algorithm

Suppose we map our universe $[n]$ to itself via a random permutation π. Then if the number of distinct elements in a stream is d, we expect $d/2^i$ of them to be divisible by 2^i after applying π. This is the core idea of the following algorithm.

Define $\texttt{tz}(x) := \max\{i \mid 2^i \text{ divides } x\}$ (i.e. the number of trailing zeroes in the base-2 representation of x).

Algorithm DistinctElements (AMS)
Input: the data stream α.

1. *Init*: Choose a random hash function $h : [n] \to [n]$ from a 2-independent family.
2. $z \leftarrow 0$.
3. *Process*(x):
4. If $\texttt{tz}(h(x)) > z$: $z \leftarrow \texttt{tz}(h(x))$.

Output: $\hat{d} \leftarrow 2^{z+1/2}$

Lemma: The AMS algorithm is a $(3, \delta)$-estimator for some constant δ.

Proof: For $j \in [n]$, $r \geq 0$, let $X_{r,j} := [\texttt{tz}(h(j)) \geq r]$, the indicator that is true if $h(j)$ has at least r trailing 0s. Now define

$$Y_r = \sum_{j : f_j > 0} X_{r,j}$$

How is our estimate related to Y_r? If the algorithm outputs $\hat{d} \geq 2^{a+1/2}$, then we know that $Y_a > 0$. Similarly, if the output is smaller than $2^{a+1/2}$, then we know that $Y_a = 0$. We will now bound the probabilities of these events.

For any $j \in [n]$, $h(j)$ is uniformly distributed over $[n]$ (since h is 2-independent). Hence $\mathbb{E}[X_{r,j}] = 1/2^r$. By linearity of expectation, $\mathbb{E}[Y_r] = d/2^r$.

We will also use the variance of these variables – note that

$$\mathrm{Var}[X_{r,j}] \leq \mathbb{E}[X_{r,j}^2] = \mathbb{E}[X_{r,j}] = 1/2^r$$

And because h is 2-independent, the variables $X_{r,j}$ and $X_{r,j'}$ are independent for $j \neq j'$, and hence:

$$\mathrm{Var}[Y_r] = \sum_{j:f_j>0} \mathrm{Var}[X_{r,j}] \leq d/2^r$$

Now, let a be the smallest integer such that $2^{a+1/2} \geq 3d$. Then we have:

$$\Pr[\hat{d} \geq 3d] = \Pr[Y_a > 0] = \Pr[Y_a \geq 1]$$

Using Markov's inequality we get:

$$\Pr[\hat{d} \geq 3d] \leq \mathbb{E}[Y_a] = \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}$$

For the other side, let b be the smallest integer so that $2^{b+1/2} \leq d/3$. Then we have:

$$\Pr[\hat{d} \leq d/3] = \Pr[Y_{b+1} = 0] \leq \Pr[|Y_{b+1} - \mathbb{E}[Y_{b+1}]| \geq d/2^{b+1}]$$

Using Chebyshev's inequality, we get:

$$\Pr[\hat{d} < d/3] \leq \frac{\mathrm{Var}[Y_b]}{(d/2^{b+1})^2} \leq \frac{2^{b+1}}{d} \leq \frac{\sqrt{2}}{3}$$

□

The previous algorithm is not particularly satisfying – by our analysis it can make an error around 94% of the time (taking the union of the two bad events). However we can improve the success probability easily; we run t independent estimators simultaneously, and print the median of their outputs. By a standard use of Chernoff Bounds one can show that the probability that the median is more than $3d$ is at most $2^{-\Theta(t)}$ (and similarly also the probability that it is less than $d/3$).

Hence it is enough to run $\mathcal{O}(\log(1/\delta))$ copies of the AMS estimator to get a $(3, \delta)$ estimator for any $\delta > 0$. Finally, we note that the space used by a single estimator is $\mathcal{O}(\log n)$ since we can store h in $\mathcal{O}(\log n)$ bits, and z in $\mathcal{O}(\log \log n)$ bits, and hence a $(3, \delta)$ estimator uses $\mathcal{O}(\log(1/\delta) \cdot \log n)$ bits.

The BJKST Algorithm

We will now look at another algorithm for the distinct elements problem. Note that unlike the AMS algorithm, it accepts an accuracy parameter ε.

Algorithm DistinctElements (BJKST)
Input: the data stream α, the accuracy ε.

1. *Init*: Choose a random hash function $h : [n] \to [n]$ from a 2-independent family.
2. $z \leftarrow 0$, $B \leftarrow \emptyset$.
3. *Process*(x):
4. If $\mathtt{tz}(h(x)) \geq z$:
5. $B \leftarrow B \cup \{(x, \mathtt{tz}(h(x))\}$
6. While $|B| \geq c/\varepsilon^2$:
7. $z \leftarrow z + 1$.
8. Remove all (a, b) from B such that $b = \mathtt{tz}(h(a)) < z$.

Output: $\hat{d} \leftarrow |B| \cdot 2^z$.

Lemma: For any $\varepsilon > 0$, the BJKST algorithm is an (ε, δ)-estimator for some constant δ.

Proof: We setup the random variables $X_{r,j}$ and Y_r as before. Let t denote the value of z when the algorithm terminates, then $Y_t = |B|$, and our estimate $\hat{d} = |B| \cdot 2^t = Y_t \cdot 2^t$.

Note that if $t = 0$, the algorithm computes d exactly (since we never remove any elements from B, and $\hat{d} = |B|$). For $t \geq 1$, we say that the algorithm *fails* iff $|Y_t \cdot 2^t - d| > \varepsilon d$. Rearranging, we have that the algorithm fails iff:

$$\left| Y_t - \frac{d}{2^t} \right| \geq \frac{\varepsilon d}{2^t}$$

To bound the probability of this event, we will sum over all possible values $r \in [\log n]$ that t can take. Note that for *small* values of r, a failure is unlikely when $t = r$, since the required deviation $d/2^t$ is large. For *large* values of r, simply achieving $t = r$ is difficult. More formally, let s be the unique integer such that:

$$\frac{12}{\varepsilon^2} \leq \frac{d}{2^s} \leq \frac{24}{\varepsilon^2}$$

Then we have:

$$\Pr[\text{fail}] = \sum_{r=1}^{\log n} \Pr\left[\left|Y_r - \frac{d}{2^r}\right| \geq \frac{\varepsilon d}{2^r} \wedge t = r\right]$$

After splitting the sum around s, we bound small and large values by different methods as described above to get:

$$\Pr[\text{fail}] \leq \sum_{r=1}^{s-1} \Pr\left[\left|Y_r - \frac{d}{2^r}\right| \geq \frac{\varepsilon d}{2^r}\right] + \sum_{r=s}^{\log n} \Pr\left[t = r\right]$$

Recall that $\mathbb{E}[Y_r] = d/2^r$, so the terms in the first sum can be bounded using Chebyshev's inequality. The second sum is equal to the probability of the event $[t \geq s]$, that is, the event $Y_{s-1} \geq c/\varepsilon^2$ (since z is only increased when B becomes larger than this threshold). We will use Markov's inequality to bound the probability of this event.

Putting it all together, we have:

$$\begin{aligned}\Pr[\text{fail}] &\leq \sum_{r=1}^{s-1} \frac{\operatorname{Var}[Y_r]}{(\varepsilon d/2^r)^2} + \frac{\mathbb{E}[Y_{s-1}]}{c/\varepsilon^2} \leq \sum_{r=1}^{s-1} \frac{d/2^r}{(\varepsilon d/2^r)^2} + \frac{d/2^{s-1}}{c/\varepsilon^2} \\ &= \sum_{r=1}^{s-1} \frac{2^r}{\varepsilon^2 d} + \frac{\varepsilon^2 d}{c 2^{s-1}} \leq \frac{2^s}{\varepsilon^2 d} + \frac{\varepsilon^2 d}{c 2^{s-1}}\end{aligned}$$

Recalling the definition of s, we have $2^s/d \leq \varepsilon^2/12$, and $d/2^{s-1} \leq 48/\varepsilon^2$, and hence:

$$\Pr[\text{fail}] \leq \frac{1}{12} + \frac{48}{c}$$

which is smaller than (say) $1/6$ for $c > 576$. Hence the algorithm is an $(\varepsilon, 1/6)$-estimator.

□

As before, we can run $\mathcal{O}(\log \delta)$ independent copies of the algorithm, and take the median of their estimates to reduce the probability of failure to δ. The only thing remaining is to look at the space usage of the algorithm.

The counter z requires only $\mathcal{O}(\log \log n)$ bits, and B has $\mathcal{O}(1/\varepsilon^2)$ entries, each of which needs $\mathcal{O}(\log n)$ bits. Finally, the hash function h needs $\mathcal{O}(\log n)$ bits, so the total space used is dominated by B, and the algorithm uses $\mathcal{O}(\log n/\varepsilon^2)$ space. As before, if we use the median trick, the space used increases to $\mathcal{O}(\log \delta \cdot \log n/\varepsilon^2)$.

(TODO: include the version of this algorithm where we save space by storing $(g(a), \texttt{tz}(h(a)))$ instead of $(a, \texttt{tz}(h(a)))$ in B for some hash function g as an exercise?)