Streaming IV : Counting Distinct Elements[1]

- In the next few lectures, we will concern ourselves with *distinct* elements in a stream. For today, we will consider insertion only streams, and each update is of the form (i, c) with $c = 1$. The object is to estimate the number of distinct elements, or put in the jargon of frequencies, the *number* of elements with $\mathbf{f}_i > 0$. This quantity is also called the 0th-frequency moment $F_0 = \sum_{i=1}^{n} \mathbf{f}_i^0$ with the convention that $0^0 = 0$. Note that, then, the sum contains a one for every element with $\mathbf{f}_i > 0$.

  The algorithm we will present today may as well be called the first modern[2] streaming algorithm. It is due[3] to Phillipe Flajolet and Nigel Martin from 1985. A slightly modified version of their algorithm and a different analysis was presented in the famous Alon-Matias-Szegedy paper. Our presentation will (very slightly) differ from theirs, because I will present it in a way that I find it most convenient to think about what's going on.

- *The Main Idea and Intuition.* Throughout, we use d to denote the number of distinct elements, and let ℓ be the non-negative integer such that $2^\ell \le d < 2^{\ell+1}$, that is, $\ell = \lfloor \lg d \rfloor$. We will use $L := \lceil \lg n \rceil$, and for now, assume we know n, and therefore, know L.

  Think of maintaining L different hash functions $h_1, \ldots, h_L : [n] \to \{0, 1\}$ such that for any $1 \le i \le L$, we have the property that for any $e \in [n]$, $\mathbf{Pr}[h_i(e)] = \frac{1}{2^i}$. Again, for the sake of intuition, for now assume h_i's are *truly random*, that is, for any $1 \le i \le L$, the random variables $\{h_i(e) : e \in [n]\}$ are mutually independent.

  The main observation is this : if there are d distinct elements, and each element was being hashed to 1 with probability $1/d$, then with constant probability one of them will hash to 1. And if the elements were being hashed to 1 with probability $\frac{1}{2^j d}$, then the probability of hashing to 1 is $\approx \frac{O(1)}{2^j}$, and thus, we don't expect any 1s from these "higher" hash functions. This leads to the main idea of the algorithm : keep L counters, setting the ith counter if any element which arrives hashes h_i to 1. At the end, return the (reciprocal of the probability associated) largest counter which is 1. Let us precisely state this algorithm, and then later on we will see how to improve its implementation.

```
1: procedure BASIC-PC:
2:     Choose $L = \lceil \lg n \rceil$ hash functions $h_1, \ldots, h_L$. ▷ $h_i(e) = 1$ w.p. $\frac{1}{2^i}$, for all $e \in [n]$.
3:     Maintain $L$ counter-"bits" $C[1 : L]$, initialized to $0$.
4:     for when element $e$ arrives do:
5:         If $h_i(e) = 1$, set $C[i]$ to $1$, for all $1 \le i \le L$.
6:     Let $Z$ be the largest index $i$ with $C[i] = 1$.
7:     return est $\leftarrow 2^Z$. ▷ The probability associated with a counter $i$ is $\frac{1}{2^i}$.
```

- *Analysis.*

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th April, 2021*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

[2] Yes, for this streaming module, we seem to be have gone reverse chronological, but this was not my intention at all.

[3] *Probabilistic Counting Algorithms for Data Base Applications.* Philippe Flajolet and G. Nigel Martin. Journal of Computer and System Sciences, vol 31(2), 1985, pp. 182-209

Theorem 1 (Basic PC Analysis.)**.** With probability $\geq \frac{3}{5}$, the estimate returned by BASIC-PC satisfies $\frac{d}{4} \leq \mathsf{est} \leq 8d$.

Proof. For any fixed $1 \leq i \leq L$, what is the probability $C[i] = 1$? It is 1 if any of the d distinct elements e satisfies $h_i(e) = 1$. Since we have assumed independence, we get

$$\mathbf{Pr}\left[C[i] = 0\right] = \mathbf{Pr}\left[\bigwedge_{e\in[n]} h_i(e) = 0\right] = \left(1 - \frac{1}{2^i}\right)^d$$

Therefore, the probability

$$\mathbf{Pr}[C[\ell-1] = 0] = \left(1 - \frac{1}{2^{\ell-1}}\right)^d \underbrace{\leq}_{\text{since } (1-t)\leq e^{-t} \text{ for any } t} e^{-\frac{d}{2^{\ell-1}}} \underbrace{\leq}_{\text{since } d\geq 2^\ell} \frac{1}{e^2} \tag{1}$$

On the other hand, for any j, the probability

$$\mathbf{Pr}[C[j] = 1] = 1 - \left(1 - \frac{1}{2^j}\right)^d \underbrace{\leq}_{\text{since } (1-t)^d\geq 1-dt \text{ for } t\geq 0} \frac{d}{2^j}$$

giving us,

$$\mathbf{Pr}[\exists j \geq \ell + 4 \ : \ C[j] = 1] \underbrace{\leq}_{\text{union bound}} \sum_{j\geq \ell+4} \frac{d}{2^j} \underbrace{\leq}_{\text{geometric series}} \frac{d}{2^{\ell+3}} \underbrace{\leq}_{\text{since } d\leq 2^{\ell+1}} \frac{1}{4} \tag{2}$$

Therefore, by applying the union bound on (1) and (2), we get that with probability $1 - \left(\frac{1}{e^2} + \frac{1}{4}\right) \geq \frac{3}{5}$, we get that the following events occur

$$\{C[\ell-1] = 1 \text{ and } C[j] = 0 \text{ for } j \geq \ell + 4\} \Rightarrow \{\ell-1 \leq Z \text{ and } Z \leq \ell+3\} \underbrace{\Rightarrow}_{\text{using } 2^\ell\leq d\leq 2^{\ell+1}} \{\frac{d}{4} \leq \mathsf{est} \leq 8d\}$$

□

Remark: *If you are "worried" about $1/4$ in the left inequality and 8 in the right inequality, then simply multiply your estimate by $1/\sqrt{2}$ to get a more symmetric answer. Whatever you do, this is a constant factor approximation, and indeed, can't be any better than 2 since we return a power of 2. The "error probability" can be reduced by the "median trick" : take $O(\ln(1/\delta))$ copies and take the median. You will be guaranteed the answer in the desired range with $1 - \delta$ probability. What's the best constant you can get?*

- ***The Better Implementation a la Flajolet-Martin.*** Apart from the fact that we used completely random functions, the above analysis was also wasteful in maintaining the L counters and L *different* hash-functions. The observation to save space is notice that the above algorithm can be implemented by using ***one*** hash function and ***one*** counter. This is usually the way this algorithm is presented, but I feel there is merit in teasing these two ideas out.

Note that the h_i's we needed have a very specific structure. And indeed, you probably can see how a single hash function gives this. Consider a hash function $h : [n] \to [n]$, and consider the binary expansion of $h(e)$. Assuming, for the moment that n is a power of 2, we see that $h(e)$ is even with probability 1, divisible by 4 with probability $1/4$, and so on. In particular, if we define $h_i(e)$ to be 1 if the "last" i bits of $h(e)$ are 0, then we get the behavior we needed from the $L = \lg n$ different hash functions. With a single one!

The second observation is that although we evaluate $h_i(e)$ for all $1 \le i \le L$ in the BASIC-PC algorithm, all we *really* care in the end is the *largest-indexed* counter which sets to 1. *Since nothing is deleted*, once the largest index is set to some r, why even bother about the smaller indexed counters? This allows us to get away by maintaining only one counter. With this, we can present the way the probabilistic counting algorithm ala Flajolet-Martin is usually presented.

```
1: procedure FLAJOLET-MARTIN:
2:     Choose $h : [n] \to [2^L]$ from a strongly universal hash family[a].▷ $L = \lceil \lg n \rceil$.
3:     ▷ Flajolet-Martin assume the existence of random hash function. The algorithm with pairwise-independent hash functions was first considered by Alon-Matias-Szegedy.
4:     Maintain a single counter $Z$ initialized to 0.
5:     for when element $e$ arrives do:
6:         $r(e) \leftarrow$ number of trailing 0s in the binary representation of $h(e)$.
7:         if $r(e) \ge Z$ then:
8:             $Z \leftarrow r(e)$.
9:     return est $\leftarrow 2^Z$.
```

[a]We need : for any $h(e)$ is uniformly distributed among $[2^L]$, and $h(e)$ and $h(e')$ for any two $e \neq e'$ are independent. This is called pairwise independence.

- *Analysis using pairwise independence.* As mentioned above, if $h : [n] \to [2^L]$ were a truly random function, that is, if the random variables indicating the number of trailing zeros of $h(e)$ and $h(e')$ and $h(e'')$ for different elements were mutually independent, then the previous analysis would have gone through. We now show how just pairwise independence is sufficient. In particular, we show

Theorem 2 (Basic PC Analysis.)**.** With probability $> \frac{1}{2}$, the estimate returned by FLAJOLET-MARTIN satisfies $\frac{d}{8} \le \text{est} \le 4d$.

  – For any integer r, let $X_{e,r}$ be the indicator variable that $h(e)$ has $\ge r$ trailing zeros. Let $Y_r = \sum_{e \in D} X_{e,r}$ denote the sum of $X_{e,r}$ over the distinct elements of the stream. We have used D to denote this set, with $|D| = d$. The final value of the counter Z is related to Y_r as follows:

$$\{Y_r \ge 1\} \Leftrightarrow \{Z \ge r\} \quad \text{and} \quad \{Y_r = 0\} \Leftrightarrow \{Z \le r - 1\} \tag{3}$$

  If $Y_r \ge 1$, then some element e will have $r(e) \ge r$ implying $Z \ge r$. On the other hand, $Y_r = 0$ implies *all* elements e have $r(e) \le r - 1$ implying $Z \le r - 1$.

  – What is the probability $X_{e,r} = 1$? That is, the probability that 2^r divides e. Since h is drawn from the strongly UHF, we get that $h(e)$ is uniformly distributed in $[2^L]$. This implies,

$\mathbf{Exp}[X_{e,r}] = \frac{1}{2^r}$, which in turn implies

$$\text{For any } 1 \le r \le L, \quad \mathbf{Exp}[Y_r] = \frac{d}{2^r} \tag{4}$$

Remark: *Therefore, we get that for any $1 \le r \le L$, the random variable $2^r Y_r$ is an unbiased estimate of d.*

– Next, we consider the variance of Y_r. To do so, we notice that Y_r is the sum of pairwise independent random variables. This pairwise independence is due to the choice of the hash family. Therefore, the variance of Y_r is the sum of the variances of $X_{e,r}$. But the variance of $X_{e,r}$ is simply $\frac{1}{2^r} \cdot \left(1 - \frac{1}{2^r}\right)$ giving us

$$\text{For any } 1 \le r \le L, \quad \mathbf{Var}[Y_r] = \frac{d}{2^r} \cdot \left(1 - \frac{1}{2^r}\right) \tag{5}$$

– The rest of the work is done by the two Russians: Chebyshev and Markov. Recall, ℓ is such that $2^\ell \le d < 2^{\ell+1}$. Now, by (3), we have $\mathbf{Pr}[Z \le \ell - 3] = \mathbf{Pr}[Y_{\ell-2} = 0]$, which in turn can be bounded as follows

$$\begin{aligned} \mathbf{Pr}[Z \le \ell - 3] = \mathbf{Pr}[Y_{\ell-2} = 0] \quad &\le \quad \mathbf{Pr}\left[\left|Y_{\ell-2} - \mathbf{Exp}[Y_{\ell-2}]\right| \ge \mathbf{Exp}[Y_{\ell-2}]\right] \\ &\underbrace{\le}_{\text{Chebyshev}} \frac{\mathbf{Var}[Y_{\ell-2}]}{\mathbf{Exp}^2[Y_{\ell-2}]} = \left(1 - \frac{1}{2^{\ell-2}}\right) \cdot \frac{1}{d/2^{\ell-2}} \underbrace{<}_{\text{since } d \ge 2^\ell} \frac{1}{4} \end{aligned}$$

Similarly, $\mathbf{Pr}[Z \ge \ell + 3] = \mathbf{Pr}[Y_{\ell+3} \ge 1]$. By Markov's inequality, we get

$$\mathbf{Pr}[Z \ge \ell + 3] = \mathbf{Pr}[Y_{\ell+3} \ge 1] \le \mathbf{Exp}[Y_{\ell+3}] = \frac{d}{2^{\ell+3}} \underbrace{<}_{\text{since } d < 2^{\ell+1}} \frac{1}{4}$$

Therefore, with probability $> \frac{1}{2}$, we get that

$$\{\ell - 2 \le Z \le \ell + 2\} \;\Rightarrow\; \{\frac{d}{8} \le 2^{\ell-2} \le 2^Z \le 2^{\ell+2} \le 4d\}$$

Exercise: *Tighten the above analysis. In particular, find the best constant c such that you can estimate something in $[d/c, cd]$ with probability $\ge 1 - \delta$. Note that for the median trick, all you need is that the probability you are an underestimate is $< 1/2$ and the probability you are an overestimate is $< 1/2$.*

• *Space Usage of* FLAJOLET-MARTIN. How much space does the algorithm take? Instead of L hash functions, there is a single hash-function which takes $O(\log n)$ bits, or $O(1)$ words. The estimate Z, on the other hand, takes only $\lg\lg n + O(1)$ bits. This is pretty impressive. The estimate of the number of distinct elements can be summarized (upto $O(1)$ factor) in $\approx \lg\lg n$ bits. At some level, one can think of this as Z maintaining the "number of significant bits" required in expressing the number of distinct items, and this number itself is $\lg n$ in size, and thus takes $\lg\lg n$ bits to write down.

Remark: *It is worth staring at the* FLAJOLET-MARTIN *algorithm again as a programmer. One would not choose a hash function, but perhaps just assume the elements are first hashed into say* 64*-bit strings before they come to you. Thus, you actually obtain* $h(e)$ *instead of* e*. Now, one can then do what are called "bit-whacking" operations to quickly evaluate* $2^{r(e)}$ *and compare with the current* 2^Z*: it involves taking complements, bit-wise ORs and bit-wise ANDs. In particular, they can be done insanely fast. And indeed, this algorithm is extremely practical. Much more so than picking the* h *from a pairwise independent hash-family.*

The analysis given above is the one given in the Alon-Matias-Szegedy paper which uses pairwise independent hash functions. Flajolet and Martin, in their original paper actually prove very powerful statements using techniques which are beyond the scope of this course (I don't understand them yet). In particular, the paper shows that if one assume h *is a truly random function (as an axiom, say), then although* est *is not an unbiased estimate,* est/ϕ *is very close to one, where* ϕ *is some irrational number* ≈ 0.77351*. Furthermore, the variance is also small, that is, the variance divided by mean square is also a small constant. Indeed, they prove that just taking the average of a bunch of trials gives a very good approximation to* d*, the number of distinct elements. For more details, I will refer you to the paper, with the warning that it is not an elementary one to read.*

The Flajolet-Martin algorithms, and its two generalizations (both of which included Flajolet as a co-author) LOGLOG *and* HYPERLOGLOG *are widely used in practice. Although the algorithms are simple to understand, their analyses are even more involved. In this offering of the course, I have made the choice of describing algorithms whose analyses I personally understand, and will not present the generalizations mentioned above. However, you should have all the tools needed to understand what these algorithms do. Indeed, just figuring what those algorithms are doing and writing about them in the parlance we have been using, could be a nice reading/writing project.*