Streaming VI : Counting Distinct Elements with Deletions[1]

- In the last two lectures, we saw algorithms to estimate the number of distinct elements in an insertion only stream. But, as is, these algorithms do not work with deletions. More precisely, every update in the stream is of the form "add e" or "delete e". We will assume for this lecture that "delete e" will only occur if there was an e to begin with. But as we see, this is not going to be strictly necessary (in case we want to handle situations when e appears -1 times). We will use d to denote the number of distinct items in the stream *at the end*. Note that in the middle there could be many more distinct elements. And this is the key problem to overcome in a sense. Jargonically, this statistic is called the L_0; the L implies that the updates allow deletion.

- One reason FLAJOLET-MARTING and BJKST don't work is that the algorithms don't remember items which hash to "smaller indexed" counters/buckets. For instance, imagine running BJKST on the stream which inserts the elements $(1, 2, 3, \ldots, n)$ but then *deletes* $(1, 2, \ldots, n - \lceil \lg n \rceil)$. At the end of the stream, the number of distinct items is $\lceil \lg n \rceil$. What does BJKST do? Well, by the time we have seen the first $n - \lceil \lg n \rceil$ elements, the value of rmin has been set such that $2^{\mathsf{rmin}} \approx \varepsilon^2 n$ since $\frac{c}{\varepsilon^2} \cdot 2^{\mathsf{rmin}}$ approximates $n - \lceil \lg n \rceil$ well. Which means, the next $\lceil \lg n \rceil$ insertions will almost surely be ignored : the chance that any one of them has $\geq$ rmin trailing zeros is $\approx \frac{1}{\varepsilon^2 n}$, and thus whp, none of them will have that rmin trailing zeros. And thus, they will be ignored. But these are the distinct elements after the deletions. And since BJKST forgets these, it can't possibly give the correct answer. Indeed, after all the deletions, all the buckets will be empty, and the estimate would be 0.

- However, the basic idea is still the same. If there are d distinct items, and we sample at rate $1/d$, then with constant probability we expect to see one item. Thus, an estimate of d can be found by taking the reciprocal of the smallest sampling probability at which we see at least one item.

  For this lecture, let us stick to the following "separation problem" : given a parameter t decide whether $d \leq t$ or $d > 2t$. More precisely, if $d \leq t$, then the algorithm asserts this with high probability, and if $d > 2t$, the same guarantee holds. When $t < d \leq 2t$, no guarantee is made on the algorithm. If such a subroutine exists, then we get a 2-approximation to d, with high probability. Indeed, run this subroutine in parallel on the t's which are the $O(\lg n)$ powers of 2, $\{1, 2, 4, 8, \ldots, 2^{\lfloor \lg n \rfloor}\}$, and find the *smallest* $t_* = 2^i$ such the algorithm asserts $d \leq t_*$. Return t^* as your answer. Note that if $2^i \leq d < 2^{i+1}$, then whp the algorithm says "$d < t$" for $t = 2^{i+1}$ and says "$d > 2t$" for $t = 2^{i-1}$. Thus, whp, the algorithm returns 2^i or 2^{i+1}, which are 2-approximations to d.

  To solve the separation problem, the algorithm up front samples (via hash functions) $S \subseteq [n]$ with probability $\approx \frac{1}{t}$ each. In the stream, the algorithm only focuses on elements in S and decides if $\mathbf{f}_i = 0$ for all $i \in S$. This corresponds to just maintaining counts on the *number* of updates[2]. The point is, if $d < t$, then there is a constant probability that all $\mathbf{f}_i$'s in S are 0, while if $d \geq 2t$, this probability is strictly smaller by a constant amount. One can use this gap by taking repeated samples, and then using Chernoff bounds. Details follow.

- *Algorithm.*

[1]*Lecture notes by Deeparnab Chakrabarty. Last modified : 15th 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]in the case the stream isn't allowed to delete an element which is not there

```
1:  procedure SeparationProblem($t, \delta$):
2:      Choose $k$ hash functions $h_1, \ldots, h_k : [n] \to [4t]$ from a pairwise independent hash
    family. ▷ $k = O(\ln(1/\delta))$.
3:      For $1 \le i \le k$, $S_i := \{e \in [n] : h_i(e) = 0\}$. ▷ This can be computed on the fly.
4:      Maintain counters $C_i$ for $1 \le i \le k$.
5:      for Update add/delete $e$ do:
6:          for $i = 1$ to $k$ do:
7:              if $e \notin S_i$ then:
8:                  Do nothing.
9:              else:
10:                 increment/decrement counter $C_i$ depending on whether add or delete.
11:     Let $N$ be the number of counters which are positive at the end of the stream.
12:     If $N > \frac{7k}{24}$, return BIG, that is, $d \ge 2t$. Otherwise, return SMALL.
```

Theorem 1. If $d \le t$, then the above algorithm returns SMALL with probability $\ge 1 - \delta$. If $d > 2t$, then the above algorithm returns BIG with probability $\ge 1 - \delta$.

Proof. Let $D \subseteq [n]$ be the set of elements which have $\mathbf{f}_i > 0$ at the end of the stream. For $1 \le i \le k$, let $\mathcal{E}_i$ be the event that the set S_i intersects D. That is,

$$\text{For } 1 \le i \le k, \;\; \mathcal{E}_i := \{S_i \cap D \neq \emptyset\}$$

We can upper bound and lower bound the probability of $\mathcal{E}_i$ as follows.

– The probability of $\mathcal{E}_i$ can be upper bounded by the union bound. For any element $e \in D$, the probability $e \in S_i$ is $= \frac{1}{4t}$ by the property of the pairwise hash family. And therefore union bound gives us

$$\mathbf{Pr}[\mathcal{E}_i] \le \frac{d}{4t} \tag{1}$$

– To *lower bound* the probability of $\mathcal{E}_i$, we use another probabilistic inequality which we haven't seen so far. It says given event $A_1, \ldots, A_n$, the probability of the union is *at least*

$$\mathbf{Pr}[\bigcup_{j=1}^{n} A_j] \ge \sum_{j=1}^{n} \mathbf{Pr}[A_j] - \sum_{1 \le j < \ell \le n} \mathbf{Pr}[A_j \cap A_\ell] \tag{Inclusion-Exclusion}$$

This can be used to lower bound $\mathcal{E}_i$ by noting that $\mathcal{E}_i$ is the union of the events A_e for $e \in D$ where A_e occurs if $e \in S_i$. Thus, using (Inclusion-Exclusion), we get

$$\mathbf{Pr}[\mathcal{E}_i] \ge \sum_{e \in D} \frac{1}{4t} - \sum_{e, e' \in D} \frac{1}{16t^2} \ge \frac{d}{4t} - \frac{\binom{d}{2}}{16t^2} \ge \frac{d}{4t} \cdot \left(1 - \frac{d}{8t}\right) \tag{2}$$

Therefore, we get that: if $d \le t$, $\mathbf{Pr}[\mathcal{E}_i] \le \frac{1}{4}$. On the other hand, if $6t \ge d > 2t$, then $\mathbf{Pr}[\mathcal{E}_i] \ge \frac{3}{8}$; one can see by inspecting (2). What about the case when $d > 6t$? When d becomes large, (2) loses its efficacy (giving vacuous results when $d \ge 8t$). But if $d > 6t$, then a simple Chebyshev application solves the problem.

Exercise: *If $d > 6t$, prove that $\mathbf{Pr}[\mathcal{E}_i] > \frac{1}{3}$ using Chebyshev.*

Proof. Let $Z := |S_i \cap D|$. Note $\mathbf{Exp}[Z] = \frac{d}{4t} > \frac{3}{2}$, and since the function is drawn from a pairwise independent hash family, $\mathbf{Var}[Z] \leq \mathbf{Exp}[Z]$. Thus, $\mathbf{Pr}[Z = 0] \leq \mathbf{Pr}[|Z - \mathbf{Exp}[Z]| > \mathbf{Exp}[Z]] \leq \frac{1}{\mathbf{Exp}[Z]} < \frac{2}{3}$. □

Let N denote the number of counters which are positive at the end of the stream. We see that $\mathbf{Exp}[N] = \sum_{i=1}^{k} \mathbf{Pr}[\mathcal{E}_i]$. Thus, when $d \leq t$, $\mathbf{Exp}[N] \leq \frac{k}{4}$ and when $d > 2t$, $\mathbf{Exp}[N] > \frac{k}{3}$. A Chernoff application gives

$$\text{If } d \leq t, \quad \mathbf{Pr}\left[N > \frac{7k}{24}\right] \leq e^{-C_1 k} \quad \text{and} \quad \text{If } d \geq 2t, \quad \mathbf{Pr}\left[N < \frac{7k}{24}\right] \leq e^{-C_2 k}$$

where C_1 and C_2 are two constants. Thus, if $k \geq C \ln(1/\delta)$ for a large enough constant C, the theorem follows. □

```
1: procedure COUNT DISTINCT WITH DELETIONS:
2:     For $t \in \{1, 2, 4, 8, \ldots, 2^{\lfloor \lg n \rfloor}\}$ run SEPARATIONPROBLEM($t, \eta$) with $\eta = \frac{\delta}{\lg n}$.
3:     Return the smallest $t$ such that SEPARATIONPROBEM($t, \eta$) returns SMALL.
```

Theorem 2. COUNT DISTINCT WITH DELETIONS returns an estimate t_* with $\frac{d}{2} \leq t_* \leq 2d$.

Proof. Let i be such that $2^i \leq d \leq 2^{i+1}$. Thus, when $t \leq 2^{i-1}$, we have $d \geq 2t$. And thus, with probability $1 - \eta$, SEPARATIONPROBLEM(t) will return BIG. Therefore, the probability there exists some $t \leq 2^{i-1}$ such that SEPARATIONPROBLEM returns SMALL is $\leq i \cdot \eta$. Similarly, when $t = 2^{i+1}$ we have $d \leq t$, and thus with probability $1 - \eta$, SEPARATIONPROBLEM(2^{i+1}) returns SMALL. In sum, the estimate we return $t_* \in \{2^i, 2^{i+1}\}$ with probability $1 - (i+1)\eta \geq 1 - \delta$ since $i \leq \lg_2 n$. □

- *Space and Time.* The total space usage in the SEPARATIONPROBLEM algorithm is dominated by the k hash functions and counters, where $k = O(\lg \lg n)$. In the COUNT DISTINCT WITH DELETIONS, everything gets multiplied by $O(\lg n)$ because that's the number of t's. So, if the hash functions itself take $O(\log n)$ bits, then we get $O(\log^2 n \log \log n)$ bits. The total time per update in the SEPARATION problem is also $O(k) = O(\lg \lg n)$, and this is also multiplied by $O(\lg n)$ in COUNT DISTINCT WITH DELETIONS. In general, everything is multiplied by $O(\log n)$ as compared to the insertion only setting.

Remark: *How do we improve the factor 2? Note that there was nothing sacrosanct about the choice of 2 in the separation problem. If we could separate between $d \leq t$ and $d > (1+\varepsilon)t$, then we would get an $(1+\varepsilon)$-approximation. And indeed, if we had truly random hash functions, then the same algorithm as above would work, and the analysis would not be too difficult. The number k would get an extra $\frac{1}{\varepsilon^2}$. If we don't assume truly random hash-functions, then we can also do away with t-wise random hash-functions with $t \approx \lg(1/\varepsilon)$. A hash function is t-wise random, if* *for any t-distinct elements, their hashes are mutually independent. Such hash functions exist with space complexity $O(t \lg n)$.*

Remark: *How do we design algorithms without assuming $\mathbf{f}_i \geq 0$? Well, it boils down to the fact whether we can distinguish between the case all $\mathbf{f}_i = 0$ or not. This can be done using a randomized algorithm, and perhaps will appear in the problem set.*