CS5234 Project: Counting Distinct Elements

Group 9
Ritu Raj (A0268438N), Atul Acharya (A0248963M), Kavishvaran Srinivasan (A0304682E)

January 29, 2025

1 Introduction

The distinct elements problem is a long studied problem that started by Flajolet and Martin in 1983. The formal problem definition is to determine the *zeroth-frequency moment* of a data stream $\sigma = a_1, a_2, \ldots, a_m$, where m is the length of the stream and each element a_i takes a value in $[n]$. This moment is denoted by F_0. More formally, $F_0 = \sum_{i=1}^{n} f_i^0$ assuming $0^0 = 0$. To solve this problem exactly requires $\Omega(n)$ bits of space since we would need to store all elements of the stream. Therefore, we are interested in the approximate version, estimating a value $\tilde{F_0}$ such that $(1-\varepsilon)F_0 \le \tilde{F_0} \le (1+\varepsilon)F_0$ for some small $\varepsilon \in (0,1)$. However, solving the approximate version of the problem deterministically also require $\Omega(n)$ bits of space since enough information from the stream needs to be stored to approximate with low error. Therefore, to develop algorithms that that are sub-linear in terms of space, randomization techniques need to be used.

The work of Flajolet and Martin (FM algorithm)[1] was the first instrumental work for the approximate distinct elements problem. Many algorithms build up on the ideas of this algorithms [2, 3, 4]. The space optimality guarantees are known for this problem and was proven to be $\Omega(\varepsilon^{-2} + \log n)$[5]. In this project, we explore two algorithms "Distinct Elements in Streams: An Algorithm for the (Text) Book"[6] which is a sampling based algorithm and "An Optimal Algorithm for the Distinct Elements Problem"[4] which is a hashing based algorithm. The latter is optimal in space usage $\mathcal{O}(\varepsilon^{-2} + \log n)$, update time $\mathcal{O}(1)$ and query time $\mathcal{O}(1)$.

This project begins by discussing a basic estimator that does not rely on hash functions. It then analyzes the state-of-the-art algorithm, with a particular emphasis on the analysis of balls and bins under limited independence. Additionally, it provides a detailed explanation of the approximation guarantees of the optimal algorithm and explores optimization techniques aimed at improving space efficiency and reducing update time.

2 Preliminaries

The aim of the algorithm is to output an estimate of F_0 or the number of distinct elements in a stream, which is denoted by $\tilde{F_0}$. We use m to denote the length of the stream and $[n]$ to denote the universe or the possible indices that may appear in the stream, given by $\{1, 2, \ldots, n\}$. $C = (1 \pm \mathcal{O}(\varepsilon))D$, for real valued $C, D, \varepsilon \ge 0$ denotes that C takes values from $(1-\epsilon)D$ to $(1+\varepsilon)D$. At any point $t \in [m]$ of the stream, $I(t)$ denotes the set of indices $i_1, i_2, ..., i_t$ seen so far, and $F_0(t) = |I(t)|$, similarly $\tilde{F_0(t)}$ is the estimate of $F_0(t)$ after t stream updates. The hash functions used are not completely independent, but are picked from a k-universal hash family for some integer k. This allows for tighter bounds, while maintaining sufficient independence. Formally, a k-universal family of hash functions from X to Y is denoted by $\mathcal{H}_k(X, Y)$, and a k-wise independent hash function is a hash function chosen randomly from this family.

3 Main Content

3.1 F0 estimation algorithm using sampling strategy and basic probability

In this section we try to understand the high level overview of "Distinct Elements in Streams: An Algorithm for the (Text) Book"[6] and its approximation guarantees. The algorithm uses a simple sampling strategy. It maintains a set $\mathcal{X}$ of samples such that each element seen is independently included in $\mathcal{X}$ with equal probability and later adjusts the sampling rate p to ensure $|\mathcal{X}| \leq \text{thresh}$.The algorithm uses $O\left(\frac{1}{\varepsilon^2} \cdot \log n \cdot \left(\log m + \log \frac{1}{\delta}\right)\right)$ space in the worst case.

Algorithm 1 F_0**-Estimator**

Input: Stream $A = \langle a_1, a_2, \ldots, a_m \rangle$, ε, δ

```
1: Initialize $p \leftarrow 1$, $\mathcal{X} \leftarrow \emptyset$, thresh $\leftarrow \lceil \frac{12}{\varepsilon^2} \log\left(\frac{8m}{\delta}\right) \rceil$
2: for $i = 1$ to $m$ do
3:     $\mathcal{X} \leftarrow \mathcal{X} \setminus \{a_i\}$
4:     With probability $p$, $\mathcal{X} \leftarrow \mathcal{X} \cup \{a_i\}$
5:     if $|\mathcal{X}| = \text{thresh}$ then
6:         Throw away each element of $\mathcal{X}$ with probability $\frac{1}{2}$
7:         $p \leftarrow \frac{p}{2}$
8:         if $|\mathcal{X}| = \text{thresh}$ then
9:             Output $\perp$
10:        end if
11:    end if
12: end for
13: Output $\frac{|\mathcal{X}|}{p}$
```

The algorithm ensures correctness by maintaining probabilities: Every distinct element of the stream is in $\mathcal{X}$ with probability $p \geq \frac{\text{thresh}}{4F_0}$. This requires bounding two events:

- **Error**: The output is not in the range $[(1-\varepsilon)F_0, (1+\varepsilon)F_0]$.
- **Fail**: The algorithm outputs $\perp$.

Using the union bound:

$$\Pr[\text{Error}] \leq \Pr[\text{Fail}] + \Pr[\text{Error} \cap \neg\text{Fail}].$$

To prove this we will use some claims and their subsequent proofs.
Claim 1: $\Pr[\text{Fail}] \leq m\left(\frac{1}{2}\right)^{\text{thresh}} \leq \frac{\delta}{8}$
Proof: Fail occurs if $|\mathcal{X}| = \text{thresh}$ and no elements can be removed. The probability of keeping an element is $\frac{1}{2}$. Summing over all m iterations, we get:

$$\Pr[\text{Fail}] \leq m\left(\frac{1}{2}\right)^{\text{thresh}} \leq \frac{\delta}{8}.$$

Claim 2: $\Pr[\text{Error} \cap \neg\text{Fail}] \leq \frac{\delta}{2}$

Proof: Consider a modified algorithm that never fails (i.e., Fail $= \perp$).
Define **Error**$_2$: The modified algorithm does not return a value in the range $[(1-\varepsilon)F_0, (1+\varepsilon)F_0]$. Then:

$$\Pr[\text{Error} \cap \neg\text{Fail}] \leq \Pr[\text{Error}_2].$$

Bounding $\Pr[\text{Error}_2]$: Define the event **Bad**: The final sampling probability p drops below $\frac{\text{thresh}}{4F_0}$. We will later use Chernoff bounds in subsequent claims to show:

$$\Pr[\text{Error}_2] \leq \Pr[\text{Bad}] + \Pr[\text{Error}_2 \mid \neg\text{Bad}] \leq \frac{\delta}{2}.$$

Claim 3: During the execution of the algorithm, the following loop invariant holds:
Each element a_i in $S_j = \{a_1, a_2, \ldots, a_j\}$,first j elements ,belongs to $\mathcal{X}$ independently with probability p.

Proof by induction:
Base Case: Since thresh > 1, for the first element a_1, it is included in $\mathcal{X}$ with probability p.
Inductive steps:

- If already in $\mathcal{X}$, it is removed and re-added with probability p.
- If not in $\mathcal{X}$, it is added with probability p.

After thresholding, p is halved, and the process continues, maintaining the invariant.

Claim 4: $\Pr[\text{Error}_2] \leq \frac{\delta}{2}$

Proof: We decompose $\Pr[\text{Error}_2]$ based on the value of p at the end of modified algorithm in Claim 2

Bounding $\Pr[\text{Bad}]$ and $\Pr[\text{Error}_2]$ will yield $\Pr[\text{Error}_2] \leq \frac{\delta}{2}$. Detailed proofs in Section A.1.

3.2 Hash based algorithm for F_0

From here on we explore F_0 estimation algorithms that use hashing based approaches. Most of the works build up on the works of the FM algorithm[1]. More specifically we will analyze the algorithm proposed by Kane et al in. their paper "An Optimal Algorithm for the Distinct Elements Problem" [4]. This algorithm draws inspiration from "Counting Distinct Elements in a Data Stream"[3] and maintains K counters $C_1, C_2, \ldots C_K$, where $K = \varepsilon^{-2}$ and uses a 2 universal hash function $h \in \mathcal{H}_2([n],[n])$ to transform the input elements of the stream to a new value in $[n]$ and computes their least significant bit $lsb(\cdot)$. Let, for now, assume that the algorithm has access to a perfectly random hash function $g \in \mathcal{H}_{[n]}([n],[K])$ to map each element of the stream to one of the counters maintained by the algorithm. The counters store the max value of $lsb(h(x))$ for all elements $x \in \sigma$. According to Bar-Yossef et al. if we know a $R \approx F_0/K$, then we can get a $(1 \pm \varepsilon)$-approximation by modeling the problem as a balls and bins problem. More specifically, let $A = \{x \in \sigma \,|\, \text{lsb}(h(x)) > R\}$ and $A = \Theta(K)$, then we can estimate $F_0 = 2^R \cdot |A|$ with a $(1 \pm \varepsilon)$ error with high probability. Let X_i denote the event counter C_i stores a value $> R$ and let $X = \sum_{i=1}^{K} X_i$. Then using the expectation of balls and bins, get $\mathbb{E}[X] = K(1-(1-\frac{1}{K})^{|A|})$. Then we can compute $|A| = \frac{\log(1-K/X)}{\log(1-1/X)}$ with $(1 \pm \varepsilon)$ error[3].

The space usage of this algorithm comes from K counters and the 2 universal hash function h (Ignoring the perfectly random hash function g). Therefore, the total space usage of this is given by $\mathcal{O}(\varepsilon^{-2}\log\log n + \log n)$ bits. To improve the space usage, they maintain the value of R such that $R \approx F_0/K$ and the counters C_i instead store a value $\max_{x:g(x)=i}(\max(\text{lsb}(h(x)) - R, 0))$. Since the counters store a value close to R, then by using this approach, the space usage per counter is $\mathcal{O}(1)$. Therefore, the total space usage is $\mathcal{O}(\varepsilon^{-2} + \log)$.

3.3 Balls and Bins under Limited Independence

The *RoughEstimator* for F_0, as seen, has parallels to the well know balls and bins problem in probability theory. The only issue of *RoughEstimator* is the use of the perfect hash function g which requires $\mathcal{O}(n\log n)$ bits of space. Using a lesser independent hash function reduces the space at the cost of approximation guarantees. To be able to use a hash function that operates on limited independence, it suffices to show that the balls and bins analysis holds using limited independence.

Let X' denote the number of bins hit by at least 1 ball using a $\mathcal{O}(k)$ universal hash function and let X denote the number of bins hit by at least one ball using a perfectly random hash function. We want the probability of the event $|X' - \mathbb{E}[X]| \le \varepsilon\mathbb{E}[X]$ to be very high. This paper[4] actually showed that we can bound this quantity using a $2(k+1)$ universal hash function where $k = c\log(K/\varepsilon)/\log\log(K/\varepsilon)$ for a sufficiently large constant $c > 0$. More formally, they show that using such a hash function, the following upper bounds can be established $|\mathbb{E}[X'] - \mathbb{E}[X]| \le \varepsilon\mathbb{E}[X]$ and $Var[X'] \le Var[X] + \varepsilon^2$. These bounds are then used to show that $\mathbb{P}[|X' - \mathbb{E}[X]| \le 8\varepsilon\mathbb{E}[X]] \ge 4/5$ which imply we can use a $2(k+1)$ universal hash function in the *RoughEstimator*. In this write up, we explore the first part of Lemma 2 and Lemma 3 and we do not cover the variance bound.

To show $|\mathbb{E}[X'] - \mathbb{E}[X]| \le \varepsilon\mathbb{E}[X]$, we can use the concept of approximate inclusion-exclusion principle. Consider two hash functions $g \in \mathcal{H}_{|A|}([A],[K])$ and $g' \in \mathcal{H}_k([A],[K])$. For a given bin i and for any ball $b \in A$, let Y_b denote the event ball b is hashed into bin i using g and let Y'_b denote the event that ball b is hashed into bin i using g'. Therefore, X_i is the indicator random variable for the event $Y = Y_1 \vee Y_2 \vee \cdots \vee Y_n$ and X'_i is the indicator random variable for $Y' = Y'_1 \vee Y'_2 \vee \cdots \vee Y'_n$. Using the inclusion-exclusion principle, we get

$$\mathbb{E}[X_i] = \mathbb{P}[Y] = \sum_{t=1}^{|A|}(-1)^{t+1}\sum_{b_1,\ldots,b_t\in A}\mathbb{P}[Y_{b_1}\wedge\cdots\wedge Y_{b_t}]$$

$$\mathbb{E}[X'_i] = \mathbb{P}[Y'] = \sum_{t=1}^{|A|}(-1)^{t+1}\sum_{b_1,\ldots,b_t\in A}\mathbb{P}[Y'_{b_1}\wedge\cdots\wedge Y'_{b_t}]$$

We can choose to sum until m terms to get the following bound if m is odd. The bound arises since each term in the summation has smaller magnitude than the previous term and is of alternating sign.

$$\sum_{t=1}^{m-1}(-1)^{t+1}\sum_{b_1,\ldots,b_t\in A}\mathbb{P}[Y_{b_1}\wedge\cdots\wedge Y_{b_t}] \le \mathbb{E}[X_i] \le \sum_{t=1}^{m}(-1)^{t+1}\sum_{b_1,\ldots,b_t\in A}\mathbb{P}[Y_{b_1}\wedge\cdots\wedge Y_{b_t}]$$

$$\sum_{t=1}^{m-1}(-1)^{t+1}\sum_{b_1,\ldots,b_t\in A}\mathbb{P}[Y'_{b_1}\wedge\cdots\wedge Y'_{b_t}] \le \mathbb{E}[X_i] \le \sum_{t=1}^{m}(-1)^{t+1}\sum_{b_1,\ldots,b_t\in A}\mathbb{P}[Y'_{b_1}\wedge\cdots\wedge Y'_{b_t}]$$

To relate $\mathbb{E}[X_i]$ and $\mathbb{E}[X'_i]$ we can set $m = k$. This makes $\mathbb{P}[Y_{b_1}\wedge\cdots\wedge Y_{b_t}] = \mathbb{P}[Y'_{b_1}\wedge\cdots\wedge Y'_{b_t}]$ by the definition of k-wise independence. Now $\mathbb{E}[X_i]$ and $\mathbb{E}[X'_i]$ only differ by the difference of the upper bound and lower bound which is $\binom{A}{k}\mathbb{P}[Y_1 \wedge Y_2 \cdots \wedge Y_k] = \binom{A}{k}(\frac{1}{K})^k$.

Therefore, $|\mathbb{E}[X_i] - \mathbb{E}[X'_i]| \leq \binom{A}{k}(\frac{1}{K})^k \leq (\frac{eA}{kK})^k \leq \frac{A}{K}(\frac{e}{k})^k$. We want to upper bound this difference by $\varepsilon\mathbb{E}[X_i]$ where $\mathbb{E}[X'_i] \approx \frac{A}{K}$. To satisfy this upper bound, it suffices to have $k = c\log(1/\varepsilon)/\log\log(1/\varepsilon)$.

Upper bounding $Var[X'] \leq Var[X] + \varepsilon^2$, requires a $2(k+1)$ universal hash function and setting $k = c\log(K/\varepsilon)/\log\log(K/\varepsilon)$[4]

Deviation of X' with respect to $\mathbb{E}[X]$: Using the above analysis, we can show that $\mathbb{P}[|X' - \mathbb{E}[X]| \leq 8\varepsilon\mathbb{E}[X]] \geq 4/5$. Since the absolute difference and the set of real number form a metric space, we can use triangle inequality to bound the term $|X' - \mathbb{E}[X]| \leq |X' - \mathbb{E}[X']| + |\mathbb{E}[X] - \mathbb{E}[X']|$. The term $|\mathbb{E}[X] - \mathbb{E}[X']|$ is a constant depending on the choice of the hash function and is upper bounded by $\varepsilon\mathbb{E}[X]$. The first term $|X' - \mathbb{E}[X']|$ can be upper bounded by $(70/11)\varepsilon\mathbb{E}[X']$ with probability $\geq 4/5$ using Chebyshev's inequality. Therefore, we get $\mathbb{P}[|X' - \mathbb{E}[X]| \leq (70/11)\varepsilon(1-\varepsilon)\mathbb{E}[X] + \varepsilon\mathbb{E}[X] \leq (1 + 70/11)\varepsilon\mathbb{E}[X] \leq 8\varepsilon\mathbb{E}[X]$ with probability $\geq 4/5$. Note that the randomness arises from the term $|X' - \mathbb{E}[X']|$.

3.4 Final F_0 estimation algorithm

Algorithm 2 **F_0 Estimation Algorithm**

```
1: Set $K = 1/\varepsilon^2$.
2: Initialize $K$ counters $C_1, \dots, C_K$ to $-1$.
3: Pick random $h_1 \in H_2([n], [0, n-1])$, $h_2 \in H_2([n], [K^3])$, $h_3 \in H_k([K^3], [K])$ for $k = \Omega(\log(1/\varepsilon)/\log\log(1/\varepsilon))$.
4: Initialize $A, b, \mathrm{est} \leftarrow 0$.
5: Run an instantiation $RE$ of $RoughEstimator$.
6: Update(i):
7:     Set $x \leftarrow \max\{C_{h_3(h_2(i))}, \mathrm{lsb}(h_1(i)) - b\}$.
8:     Set $A \leftarrow A - \lceil\log(2 + C_{h_3(h_2(i))})\rceil + \lceil\log(2 + x)\rceil$.
9:     If $A > 3K$, Output FAIL.
10:    Set $C_{h_3(h_2(i))} \leftarrow x$. Also feed $i$ to $RE$.
11:    Let $R$ be the output of $RE$.
12:    If $R > 2^{\mathrm{est}}$:
13:        (a) $\mathrm{est} \leftarrow \log(R), b_{\mathrm{new}} \leftarrow \max\{0, \mathrm{est} - \log(K/32)\}$.
14:        (b) For each $j \in [K]$, set $C_j \leftarrow \max\{-1, C_j + b - b_{\mathrm{new}}\}$.
15:        (c) $b \leftarrow b_{\mathrm{new}}, A \leftarrow \sum_{j=1}^{K}\lceil\log(C_j + 2)\rceil$.
16: Estimator:
17: Define $T = |\{j : C_j \geq 0\}|$.
18: Output $\hat{F}_0 = \frac{2^b \cdot \ln(1 - T/K)}{\ln(1 - 1/K)}$.
```

Building up on the components provided earlier, the focus moves to the final F_0 estimation algorithm as proposed by Kane et al [4] that provides the best space-efficient and approximation of F_0. The algorithm initializes K counters that resemble the bins described in Section 3.2., each of them set to -1 which represents the state of a bin with 0 balls. 3 hash functions are defined, h_1 from a 2-universal hash family that maps each index to a level between 0 and log(n), h_2 and h_3 come from a 2-universal and k-universal hash family and k-universal hash family respectively, with k set according to Lemma 1. h_2 and h_3 are used to assign each index to a counter, while maintaining sufficient independence for a good approximation. A is used to track the total space utilized by all counters C_1 ,... ,C_K, b represents the current level at which our algorithm is operating at and est is the output of *Rough-Estimator* used to provide a $\theta(1)$ approximate of F_0.

For each index that arrives in the stream, level x is computed for index i, by comparing the current highest level stored by the counter it is hashed to and the difference of the least significant bit generated by $h_1(i)$ and b. A is updated to check if it exceeds the threshold for space usage by counters, and the algorithm outputs "FAIL" if it does. The counter corresponding to index i is updated to x, and R, the output of *RoughEstimator* is obtained. If there is a change, the counters are updated and b to match the new F_0 approximate and set A to the new space occupied by all counters.

To output the F_0 estimate at any point in the stream, T is computed as the number of non-empty bins or counters that have at least one index hashed to it, and use it along with b to denote the current highest level and K to compute the F_0 estimate. A brief description of how to achieve $\mathcal{O}(1)$ update time is outlined in Section A.4.

3.5 Analysis of the F_0 estimation algorithm

Space Usage: The hash functions h_1 and h_2 take up $\mathcal{O}(log(n))$ space due to their construction, while h_2 takes up $\mathcal{O}(klogK)$ space as it is from a k-universal hash family. This can be resolved to $\mathcal{O}(log^2(1/\epsilon))$ by substituting k as $c\log(\frac{1}{\epsilon})/\log\log(\frac{1}{\epsilon})$ and K as $\mathcal{O}(\epsilon^{-2})$. A represents the total space used by all counters, and since there are K counters, each taking a maximum value of log(n), A takes up $\mathcal{O}(log(\epsilon^{-2}log(n)))$ and upon simplification we get $\mathcal{O}(log(\epsilon^{-1}) + loglog(n))$. The counters C_1, ..., C_K take up the same space as A and is hence included within the $\mathcal{O}$ representation of space. Finally b and *est* take a maximum value of log(n) as they represent levels of bins and hence use $\mathcal{O}(loglog(n))$ space. Therefore, the total space used by this algorithm is $\mathcal{O}(\epsilon^{-2} + log(n))$.

Proof of Correctness: In this section, we prove that our algorithm is correct and provides a $(1 \pm \mathcal{O}(\epsilon))$ approximation of F_0 with high probability. Before delving into the analysis, certain terminologies, assumptions and conditions used are listed below:

1. Assumption 1: $F_0 \geq \frac{K}{32}$. (Case of small F_0 will be handled in Section 3.5)
2. Assumption 2: Without loss of generality, we assume $\frac{1}{\epsilon^2} \geq Clog(n)$ for any C and is a power of 2. If we want to run the algorithm with $\epsilon > \frac{1}{\sqrt{Clog(n)}}$ then we set ϵ as $\frac{1}{\sqrt{Clog(n)}}$, weakening the space bound by a constant factor.
3. I_b is the set of indices i such that $lsb(i) \geq b$.
4. Conditioning $E1$: $F_0(t) \leq \tilde{F}_0^{RE}(t) \leq 8F_0(t)$ with probability 1-o(1).
5. Conditioning $E2$: $K/300 \leq |I_b| \leq K/20$
6. Conditioning $E3$: I_b is perfectly hashed under h_2

The analysis is split into two parts for the sake of simplicity. The first part deals with the situation of when the algorithm outputs "FAIL" and the second part assures that the output provided is a (1 $\pm$ $\mathcal{O}(\epsilon)$) approximation with high probability.

Part 1: To show the algorithm fails with a low probability. The algorithm fails when A exceeds the threshold defined i.e. $3K$ that can be utilized by all counters. Therefore, the positions where A

changes are analyzed and it suffices to show that the values of A at these positions, also referred to as critical points, remain below $3K$.

Definition: Points t_1, t_2, ... , t_{r-1} where the output of *RoughEstimator* changes, i.e. $F_0^{RE}(t_j - 1) \neq F_0^{RE}(t_j)$ for all $j \in [r-1]$ are called critical points.

Therefore, it suffices to show A(t) $\leq 3K$ where t represents any critical point, as shown below:

$$A(t) \leq K + \sum_{i=1}^{K} \log(C_i(t) + 2) \leq K + K \cdot \log\left(\frac{\sum_{i=1}^{K} C_i(t)}{K} + 2\right)$$

The final result is obtained by removing the ceiling function, applying Jensen's inequality which states that $f(tx_1+(1-t)x_2) \leq tf(x_1)+(1-t)f(x_2)$ for some weights t and $(1-t)$, and a **convex** function f , and concavity of the logarithm function. The focus is now shifted to showing $\sum_{i=1}^{K} C_i(t) \leq 2K$. We define the random variable $X_i(t)$ taking the value $max\{-1, lsb(h_1(i)) - b\}$ and X(t) $= \sum_{i \in I(t)} X_i(t)$,where $I(t)$ is set of all indices arrived at time t. Now, $\sum_{i=1}^{K} C_i(t) \leq X(t)$. Therefore, it suffices to show that $X(t) \leq 2K$.

$$X_i(t) = \begin{cases} s \in [0, log(n) - b), & \text{with probability } 2^{-(b+s+1)} \\ log(n) - b, & \text{with probability } \frac{1}{n} \\ \text{-1}, & \text{with remaining probability} \end{cases}$$

We now bound Pr[$X(t) \leq 2K$]. By using linearity of expectation and excluding the case of -1, we get:

$$\mathbb{E}[X(t)] \leq F_0(t) \cdot (\frac{1}{n} + \sum_{s=0}^{log(n)-b+1} 2^{-(b+s+1)}) = \frac{F_0(t)}{2^b}$$

By choice of h_1, $X_i(t)$ are pairwise independent, and applying Chebyshev's inequality

$$Pr[X(t) > 2K] < \frac{F_0(t)}{2^b \cdot ((2K - F_0(t)/2^b)^2},$$

Using $E1$, the above probability is at most $1/32K$. Applying union bound for all t_j, we get X(t) $\leq$ 2K with prob. at least 1 - 1/32. Hence, the algorithm fails with a low probability of $\leq 1/32$.

Part 2: To show the output T of the algorithm is a $(1 \pm \mathcal{O}(\epsilon))$ approximation of F_0. Let's denote the regular algorithm as Alg and Alg' as Alg without the failure checking step (Step 9). Let $\hat{F_0}$ be the output of Alg'. Now, we show that $\hat{F_0}$ is $(1 \pm \mathcal{O}(\epsilon))F_0$ with high probability.

We list down certain results obtained based on the conditionings:

1. $K/256 \leq \mathbb{E}[|I_b|] \leq K/20$
2. Applying Chebyshev's inequality: $Pr[E2|E1] \geq 1 - \mathcal{O}(1/K)$ = 1 - o(1).
3. Using pairwise independence of h_2 $Pr[E3|E2] \geq 1 - \mathcal{O}(1/K)$ = 1 - o(1).
4. Based on $E2 \wedge E3$, T is a random variable that denotes the number of bins that has at least one ball using a k-wise independent hash function, K bins and $B = |I_b|$ balls.

Using Lemma 2, we obtain $T = (1 \pm 8\epsilon)(1-(1-1/K)^B)K$ with probability of 4/5 and now try to use this result to obtain $\tilde{F_0}$, hence we compute $ln(1-T/K) = ln((1-1/K)^B \pm 8\epsilon(1-(1-1/K)^B))$. From, the value obtained from $ln(1-T/K)$ and simplifying the error term by $E2$ and the fact that $ln(1+x) = \mathcal{O}(|x|)$ for x < 1/2, we get $\tilde{F_0} = B \cdot 2^b \pm \mathcal{O}(\epsilon \cdot 2^b K)$

Using $E1$, $2^b \leq 256F_0/K$ reducing the error term to $\mathcal{O}(\epsilon F_0)$ and $\mathbb{E}[B] = F_0/2^b$. Using pairwise independence of h_1 and chebyshev's inequality:

$$Pr[|B - \mathbb{E}[B]|] \geq c/\sqrt{K}] \leq \frac{\mathbb{E}[B]}{(c^2/K) \cdot \mathbb{E}[B]^2} \leq \frac{16}{c^2}$$

where c is a constant of our choice. The above probability can be made into a small constant by setting c to be a very large constant, and hence B = $(1 \pm \mathcal{O}(\epsilon))F_0/2^b$ with large constant probability. Applying $E1 \wedge E2 \wedge E3$, with $Pr[E1 \wedge E2 \wedge E3] \geq =$ 1 - o(1), we get

$$Pr[\tilde{F_0} = (1 \pm \mathcal{O}(\epsilon))F_0] \geq 3/5 - o(1)$$

Therefore, combining results of Part 1 and Part 2, our algorithm does not output "FAIL" and outputs $\tilde{F_0} = (1 \pm \mathcal{O}(\epsilon))F_0$ with probability $\geq 1 - 2/5 - o(1) - 1/32 > 11/20$

3.6 Dealing with small F_0

In this section, we deal with cases outside our previous assumption used during analysis, stating that $F_0 \geq K/32$.

Case $F_0 < 100$: We simply store the first 100 distinct indices seen during the stream, which requires $\mathcal{O}(logn)$ space.

Case $F_0 \geq 100$: We maintain additional $K' = 2K$ bits, B_1, B_2, ... , $B_{K'}$ and update the algorithm as follows: During the update stage, we add one more step assigning $B_{h_3(h_2(i))}$ as 1 and expanding h3 to have a range of 2K while performing mod K for all evaluations to keep the results consistent with the original algorithm. We now require only 2 critical points, t_0 and t_1 such that t_0 is the smallest t when $F_0(t) = K'/32$ and t_1 is the smallest t when $F_0(t) = K'/64$, if no such t_i exist set them to ∞. We now calculate $T_b(t) = |\{i : B_i(t) = 1\}|$ and $\epsilon F_0(t) = \frac{ln(1-T_B(t)/K')}{ln(1-1/K')}$. Using similar analysis as before and applying union bound over t_0 and t_1, we get that $Pr[\tilde{F_0^B}(t_i) = (1 \pm \mathcal{O}(\epsilon))F_0(t_i)] \geq 1 - 2 \cdot (1/5) - o(1) = 3/5 - o(1)$ for t_0 and t_1. Since $\tilde{F_0^B}(t)$ monotonically increases with t, we do the following: If $\tilde{F_0^B}(t) \geq K'/32 = K/16$, we output the result of regular algorithm, else we output $\tilde{F_0^B}(t)$.

4 Future Direction

As seen, the algorithm proposed by Kane et al.[4] is optimal in terms of space usage, update time and query time. Future work could focus on analyzing algorithms better or developing new ideas to achieve the same space and time complexities while reducing the constant factor approximations. This paper also proposes an algorithm for L_0 estimation where items can be deleted. From our perspective, the next steps would be to understand L_0 estimation and explore the challenges of developing algorithms when items can be removed from the stream. Additionally, another promising direction is studying how modern hardware capabilities, such as parallelism and cache hierarchies, can enhance the practical performance of these algorithms.

References

[1] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, pp. 182–209, 1985.

[2] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 137–147, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000097915452

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Randomization and Approximation Techniques in Computer Science*, J. D. P. Rolim and S. Vadhan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 1–10.

[4] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 41–52. [Online]. Available: https://doi.org/10.1145/1807085.1807094

[5] P. Indyk and D. Woodruff, "Tight lower bounds for the distinct elements problem," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, 2003, pp. 283–288.

[6] S. Chakraborty, N. V. Vinodchandran¹, and K. S. Meel, "Distinct elements in streams: An algorithm for the (text) book." Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ESA.2022.34

A Appendix

A.1 Bounding $\Pr[\textbf{Error}_2]$

For $j \in [1, m]$, let Bad_j denote the event that the j-th iteration of the for loop in modified algorithm is the first iteration where the value of p goes below $2^{-\ell}$, i.e., $p_{j-1} = 2^{-\ell}$ and $p_j = 2^{-(\ell+1)}$ where l= $\lceil \log \left(\frac{\text{thresh}}{4F_0} \right) \rceil$

Therefore,

$$\Pr[\text{Bad}] = \sum_{j=1}^{m} \Pr[\text{Bad}_j].$$

Using Chernoff bounds: For $a \in S_m$, let a be an indicator for $a \in X$. By Claim 3, $\Pr[r_a = 1] = p$ and elements are independent. $|X| = \Pr[r_a = 1]$, and $E[|X|] = p \cdot F_0$.

$$\Pr\left[|X| \geq \text{thresh}\right] \leq 2e^{-\frac{\text{thresh}^2}{3F_0 \cdot 2^l}} \leq \frac{\delta}{4m}.$$

Bounding $\Pr[\text{Error}_2 \cap \text{Bad}]$:

Define event Error_2, q as $p_m = 2^{-q}$ and $|X_m| 2^{-q} \notin [(1-\epsilon)F_0, (1+\epsilon)F_0]$.

Using similar analysis, we can bound:

$$\Pr[\text{Error2} \cap \text{Bad2}] \leq \sum_{q=0}^{\ell} 2e^{-\frac{\epsilon^2 F_0}{3 \cdot 2^q}} \leq 4e^{-\frac{\epsilon^2 \text{thresh}}{12}} \leq 4\left(\frac{\delta}{8}\right)^{\log e} \leq \frac{\delta}{4}$$

Summing these:

$$\Pr[\text{Error}_2] \leq \frac{\delta}{2}.$$

A.2 Inclusion Exclusion Principle

Inclusion Exclusion - The inclusion-exclusion principle is a counting technique to compute the size of the union of finite sets. If A, B, and C are sets, then $|A \cup B| = |A| + |B| - |A \cap B|$ and $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$. This can be generalized to the union of n sets and can be compactly represented by the following equation:

$$\left| \bigcup_{i=1}^{n} A_i \right| = \sum_{t=1}^{n} (-1)^{t+1} \sum_{\{A_1, \ldots, A_t\} \subseteq A} |A_1 \cap A_2 \cap \cdots \cap A_t| \,.$$

A.3 Bounding $|X' - \mathbb{E}[X']|$

In 3.3, we briefly mentioned how we can bound $|X' - \mathbb{E}[X']| \leq (70/11)\varepsilon\mathbb{E}[X']$ with probability $\geq 4/5$. To prove this, we require bounds for $\mathbb{E}[X']$ and $Var[X']$ in terms of A. We will consider the case $100 \leq A \leq K/20$ and $K = \varepsilon^{-2}$

Using the approximation $(1-x)^n \approx (1 - nx + \binom{n}{2}x^2)$, we get

$$\begin{aligned}
\mathbb{E}[X] &\geq K(1 - (1 - A/K + \binom{A}{2}1/K^2) \\
&\geq 1/\varepsilon^{-2}(A\varepsilon^2 - \binom{A}{2}\varepsilon^4) \\
&\geq 20A(A/(20A) - A^2/(800A^2)) \\
&\geq A(1 - 1/40) \\
&\geq (39/40)A
\end{aligned}$$

Using the previous bound $\mathbb{E}[X'] \geq (1-\varepsilon)\mathbb{E}[X] \geq (1-1/10)A \geq (9/10)A$ since $\varepsilon^{-2} \geq 100$. Given that $100 \leq A \leq K/20$ then $Var[X] < \frac{4A^2}{K}$[4]. We do not go through the proof of this but we will use it to get a upper bound for $Var[X']$. From the analysis of balls and bins under limited independence, we have $Var[X'] \leq Var[X] + \varepsilon^2 \leq \frac{4A^2}{K} + \varepsilon^2 \leq 5\varepsilon^2 A$

We can then use Chebysehv inequality to bound $\mathbb{P}[|X' - \mathbb{E}[X'] \leq (70/11)\varepsilon\mathbb{E}[X]]$.

$$\begin{aligned}
\mathbb{P}[|X' - \mathbb{E}[X'] \geq (70/11)\varepsilon\mathbb{E}[X']] &\leq \frac{Var[X']}{(70/11)^2\varepsilon^2\mathbb{E}[X']^2} \\
&\leq \frac{5\varepsilon^2 A^2}{(70/11)^2\varepsilon^2(9/10)^2\mathbb{A}^{\nvDash}} \\
&\leq 1/5
\end{aligned}$$

A.4 $\mathcal{O}(1)$ Update Time

Theorem 1 *There is a VLA data structure using $O(n + \sum_i len(C_i))$ space to store n elements, supporting worst-case $O(1)$ updates and reads, under the assumptions that:*

1. *$len(C_i) \leq w$ for all i, and*
2. *$w \geq \log(M)$.*

Here, w is the machine word size, and M is the amount of memory available to the VLA.

We briefly show how $\mathcal{O}(1)$ update time can be achieved by slight modifications. Evaluating h_1, h_2 and h_3, updating least significant bit can be performed in $\mathcal{O}(1)$ time. Updating A can be treated as a most significant bit operation, which also requires $\mathcal{O}(1)$ time. Additionally, we can perform read and write operations on the counters using constant time and asymptotic space described in Theorem 1. To ensure reporting time also falls under the same time limit, we can maintain $T = |\{j : C_j \geq 0\}|$ during updates, and thus the final $\tilde{F}$ can be computed using logarithm function, which can be expedited using a small lookup table.

Lemma 1 *RoughEstimator can be implemented with $O(1)$ worst-case update and reporting times, at the expense of only giving a 16-approximation to $F_0(t)$ for every $t \in [m]$ with $F_0(t) \geq K_{RE}$.*

The issue arises in handling a change in output of *RoughEstimators* which would require $\mathcal{O}(K)$ time. In simple terms, one way to fix it can be described as distributing the $\mathcal{O}(K)$ work across the next $\mathcal{O}(K)$ updates, since once b_{new} changes, it cannot do so for more than a constant times in the following $\mathcal{O}(K)$ updates as described in Lemma 1.

Therefore, we maintain a primary and secondary set of counters, with the secondary storage consisting of $3 \cdot 256$ copies of C_j, used for performing updates when the output of *RoughEstimator* that is *est* changes. This number $(3 \cdot 256)$ arises from the fact that if *est* changes for more than 3 times in the next $K/256$ stream updates after an initial change, the algorithm shall output "FAIL". This modification does not affect the proof of correctness as 2^b still is a constant-factor approximation of F_0 during the copying and processing stage in the secondary storage.