Algorithms for data streams

Maria Serna

Spring 2025

1. Finding frequent items
2. Counting values

Frequent items

Frequent items

- We have a stream $x_1, \dots, x_m$, where $x_i \in \Sigma$.
- This implicitly defines a frequency vector $f_1, \dots, f_n$, where $n = |\Sigma|$ with $f_1 + \cdots + f_n = m$.

Frequent items

- We have a stream $x_1, \ldots, x_m$, where $x_i \in \Sigma$.
- This implicitly defines a frequency vector $f_1, \ldots, f_n$, where $n = |\Sigma|$ with $f_1 + \cdots + f_n = m$.
- Frequent items problem:
  Given k, output the set $\{j \mid f_j > m/k\}$.

Frequent items

- We have a stream $x_1, \ldots, x_m$, where $x_i \in \Sigma$.
- This implicitly defines a frequency vector $f_1, \ldots, f_n$, where $n = |\Sigma|$ with $f_1 + \cdots + f_n = m$.
- Frequent items problem:
  Given k, output the set $\{j \mid f_j > m/k\}$.
- Frequency estimation problem:
  Process the stream to get a data structure that can provide an estimate $\hat{f}_i$ of f_i, for a given $i \in [n]$.

Frequency estimation: Naive approach

Frequency estimation: Naive approach

- Exact algorithm:

```
procedure FREQ(int n, stream s)
    int j, F[n] = 0
    while not s.end() do
        j = s.read()
        F[j]++
```

Frequency estimation: Naive approach

- Exact algorithm:

```
procedure FREQ(int n, stream s)
    int j, F[n] = 0
    while not s.end() do
        j = s.read()
        F[j]++
```

- Computes the frequency vector.

Frequency estimation: Naive approach

- Exact algorithm:

```
procedure FREQ(int n, stream s)
    int j, F[n] = 0
    while not s.end() do
        j = s.read()
        F[j]++
```

- Computes the frequency vector.
- One pass, using $O(n \log m)$ memory and $O(1)$ time per item.

Frequency Estimation: Misra-Gries algorithm

Frequency Estimation: Misra-Gries algorithm

- The algorithm has an additional parameter k.

Frequency Estimation: Misra-Gries algorithm

- The algorithm has an additional parameter k.
- Uses an associative array with n potential keys.

Frequency Estimation: Misra-Gries algorithm

- The algorithm has an additional parameter k.
- Uses an associative array with n potential keys.
- The associative array can be implemented using a balanced binary search tree.

Frequency Estimation: Misra-Gries algorithm

Frequency Estimation: Misra-Gries algorithm

```
procedure MISRA-GRIES(int n, stream s,int k)
    int A empty associative array
    while not s.end() do
        j = s.read()
        if j ∈ keys(A) then
            A[j]++
        else
            if |keys(A)| < k − 1 then
                A[j] = 1
            else
                for ℓ ∈ keys(A) do
                    A[ℓ]- -
                    if A[ℓ] == 0 then
                        remove ℓ from A
    On query a, if a ∈ keys(A), report f̂_a = A[a], else report 0.
```

Misra-Gries algorithm: cost analysis

- Only one pass.

Misra-Gries algorithm: cost analysis

- Only one pass.
- Each key requires $O(\log n)$ bits and each value $O(\log m)$ bits.
- There are at most $k-1$ key/value pairs, the total space is $O(k(\log m + \log n))$.
- The time per element is $O(k)$.

Misra-Gries algorithm: cost analysis

- Only one pass.
- Each key requires $O(\log n)$ bits and each value $O(\log m)$ bits.
- There are at most $k-1$ key/value pairs, the total space is $O(k(\log m + \log n))$.
- The time per element is $O(k)$.
- Quality of the solution?

Misra-Gries algorithm: quality analysis

Misra-Gries algorithm: quality analysis

- Let's see A as a vector with $A[i] = 0$ when $i \notin keys(A)$

Misra-Gries algorithm: quality analysis

- Let's see A as a vector with $A[i] = 0$ when $i \notin keys(A)$
- $A[j]$ is incremented only when j appears in s, so $\hat{f}_j \leq f_j$.

Misra-Gries algorithm: quality analysis

- Let's see A as a vector with $A[i] = 0$ when $i \notin keys(A)$
- $A[j]$ is incremented only when j appears in s, so $\hat{f}_j \leq f_j$.
- Whenever $A[j]$ is decremented, we decrement the values of other $k-1$ keys.
  The decrement is witnessed by k tokens including j, assuming that $A[j]$ first goes to 1 and then down to 0.

Misra-Gries algorithm: quality analysis

- Let's see A as a vector with $A[i] = 0$ when $i \notin keys(A)$
- $A[j]$ is incremented only when j appears in s, so $\hat{f}_j \leq f_j$.
- Whenever $A[j]$ is decremented, we decrement the values of other $k-1$ keys.
  The decrement is witnessed by k tokens including j, assuming that $A[j]$ first goes to 1 and then down to 0.
- Since the stream has m tokens there can be at most m/k such decrements. Therefore, $\hat{f}_j \geq f_j - m/k$.

Misra-Gries algorithm: quality analysis

- Let's see A as a vector with $A[i] = 0$ when $i \notin keys(A)$
- $A[j]$ is incremented only when j appears in s, so $\hat{f}_j \leq f_j$.
- Whenever $A[j]$ is decremented, we decrement the values of other $k - 1$ keys.
  The decrement is witnessed by k tokens including j, assuming that $A[j]$ first goes to 1 and then down to 0.
- Since the stream has m tokens there can be at most m/k such decrements. Therefore, $\hat{f}_j \geq f_j - m/k$.
- Putting all together

$$f_j - \frac{m}{k} \leq \hat{f}_j \leq f_j$$

Frequent items using Misra-Gries algorithm

Frequent items using Misra-Gries algorithm

- By the analysis, if one key j has $f_j > m/k$, $\hat{f}_j > 0$.

Frequent items using Misra-Gries algorithm

- By the analysis, if one key j has $f_j > m/k$, $\hat{f}_j > 0$.
- However, there might be elements for which $\hat{f}_j > 0$ but $f_j \leq m/k$.

Frequent items using Misra-Gries algorithm

- By the analysis, if one key j has $f_j > m/k$, $\hat{f}_j > 0$.
- However, there might be elements for which $\hat{f}_j > 0$ but $f_j \leq m/k$.
- Perform a second pass on the stream, counting exactly the frequencies of the values $i \in keys(A)$. And extracting only those verifying the property.

Frequent items using Misra-Gries algorithm

- By the analysis, if one key j has $f_j > m/k$, $\hat{f}_j > 0$.
- However, there might be elements for which $\hat{f}_j > 0$ but $f_j \leq m/k$.
- Perform a second pass on the stream, counting exactly the frequencies of the values $i \in keys(A)$. And extracting only those verifying the property.
- 2 pass algorithm, using $O(k(\log m + \log n))$ space, and $O(k)$ time per element.

1. Finding frequent items

2. Counting values

Counting the number of distinct elements

Counting the number of distinct elements

- Distinct elements problem: output $|\{j \mid f_j > 0\}|$.
- This is a simplification of the Frequent items problem:

Counting the number of distinct elements

- Distinct elements problem: output $|\{j \mid f_j > 0\}|$.
- This is a simplification of the Frequent items problem:
- In order to solve the problem using sublinear space we need to use probabilistic algorithms/data structure and some adequate notion of approximation.

An (ϵ, δ)-approximation

An (ϵ, δ)-approximation

- Let $\mathcal{A}(s)$ denote the output of a randomized streaming algorithm $\mathcal{A}$ on input s; note that this is a random variable.
- Let $\Phi(s)$ be the function that $\mathcal{A}$ is supposed to compute.

An (ϵ, δ)-approximation

- Let $\mathcal{A}(s)$ denote the output of a randomized streaming algorithm $\mathcal{A}$ on input s; note that this is a random variable.
- Let $\Phi(s)$ be the function that $\mathcal{A}$ is supposed to compute.
- $\mathcal{A}$ is a (ϵ, δ)-approximation to Φ if we have

$$Pr\left[\left|\frac{\mathcal{A}(s)}{\Phi(s)} - 1\right| > \epsilon\right] \leq \delta.$$

An (ϵ, δ)-approximation

- Let $\mathcal{A}(s)$ denote the output of a randomized streaming algorithm $\mathcal{A}$ on input s; note that this is a random variable.
- Let $\Phi(s)$ be the function that $\mathcal{A}$ is supposed to compute.
- $\mathcal{A}$ is a (ϵ, δ)-approximation to Φ if we have

$$Pr\left[\left|\frac{\mathcal{A}(s)}{\Phi(s)} - 1\right| > \epsilon\right] \leq \delta.$$

- $\mathcal{A}$ is a (ϵ, δ)-additive approximation to Φ if we have

$$Pr\left[\left|\mathcal{A}(s) - \Phi(s)\right| > \epsilon\right] \leq \delta.$$

An (ϵ, δ)-approximation

- Let $\mathcal{A}(s)$ denote the output of a randomized streaming algorithm $\mathcal{A}$ on input s; note that this is a random variable.
- Let $\Phi(s)$ be the function that $\mathcal{A}$ is supposed to compute.
- $\mathcal{A}$ is a (ϵ, δ)-approximation to Φ if we have

$$Pr\left[\left|\frac{\mathcal{A}(s)}{\Phi(s)} - 1\right| > \epsilon\right] \leq \delta.$$

- $\mathcal{A}$ is a (ϵ, δ)-additive approximation to Φ if we have

$$Pr\left[|\mathcal{A}(s) - \Phi(s)| > \epsilon\right] \leq \delta.$$

- When $\delta = 0$, $\mathcal{A}$ must be deterministic.
  When $\epsilon = 0$, $\mathcal{A}$ must be an exact algorithm.

Randomized data structures

- We need hashing and in particular hash functions selected at random from a universal hash family.

Randomized data structures

- We need hashing and in particular hash functions selected at random from a universal hash family.
- Recall that a family of functions

$$H = \{h : U \to [m]\}$$

is called a 2-universal family if, $\forall x, y \in U,\ x \neq y$,

$$\Pr_{h \in H}[h(x) = h(y)] \leq \frac{1}{m}.$$

Randomized data structures

- We need hashing and in particular hash functions selected at random from a universal hash family.
- Recall that a family of functions

$$H = \{h : U \to [m]\}$$

is called a 2-universal family if, $\forall x, y \in U,\ x \neq y$,

$$\Pr_{h \in H}[h(x) = h(y)] \leq \frac{1}{m}.$$

- A hash function can be easily selected at random from a 2-universal hash family.

Values from the binary representation

Values from the binary representation

- For an integer $p > 0$, let zeros(p) be the number of zeros at the end of the binary representation of p.

Values from the binary representation

- For an integer $p > 0$, let zeros(p) be the number of zeros at the end of the binary representation of p.

$$\text{zeros}(p) = \max\{i \mid 2^i \text{ divides } p\}.$$

Counting distinct elements

Counting distinct elements

Algorithm: Flajolet and Martin, 1983

procedure COUNT-DIF(stream s)
    Choose a random hash function $h : [n] \to [n]$
    from a universal family
    int $z = 0$
    while not $s.end()$ **do**
        $j = s.read()$
        if zeros$(h(j)) > z$ **then**
            $z =$ zeros$(h(j))$
    Return $\lfloor 2^{z+\frac{1}{2}} \rfloor$

Counting distinct elements

Algorithm: Flajolet and Martin, 1983

procedure COUNT-DIF(stream s)
  Choose a random hash function $h : [n] \to [n]$
  from a universal family
  int $z = 0$
  while not $s.end()$ **do**
    $j = s.read()$
    if zeros($h(j)$) $> z$ **then**
      $z =$ zeros($h(j)$)
  Return $\lfloor 2^{z+\frac{1}{2}} \rfloor$

- Assuming that there are d distinct elements, the algorithm computes $\max \text{zeros}(h(j))$ as a good approximation of $\log d$.

Counting distinct elements

Algorithm: Flajolet and Martin, 1983

```
procedure COUNT-DIF(stream s)
    Choose a random hash function h : [n] → [n]
    from a universal family
    int z = 0
    while not s.end() do
        j = s.read()
        if zeros(h(j)) > z then
            z = zeros(h(j))
    Return ⌊2^(z+1/2)⌋
```

- Assuming that there are d distinct elements, the algorithm computes $\max \text{zeros}(h(j))$ as a good approximation of $\log d$.
- 1 pass, $O(\log n)$ memory and $O(1)$ time per item.

Counting the number of distinct elements: Quality

Counting the number of distinct elements: Quality

- For $j \in [n]$ and $r \geq 0$, let $X_{r,j}$ be the indicator r.v. for
$$\text{zeros}(h(j)) \geq r.$$

Counting the number of distinct elements: Quality

- For $j \in [n]$ and $r \geq 0$, let $X_{r,j}$ be the indicator r.v. for
  $$\text{zeros}(h(j)) \geq r.$$
- Since $h(j)$ is uniformly distributed over the $\log n$-bit strings,

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}$$

Counting the number of distinct elements: Quality

- For $j \in [n]$ and $r \geq 0$, let $X_{r,j}$ be the indicator r.v. for
  $\text{zeros}(h(j)) \geq r$.
- Since $h(j)$ is uniformly distributed over the $\log n$-bit strings,

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}$$

- Let $Y_r = \sum_{j|f_j>0} X_{r,j}$ and let t denote the final value of z.

Counting the number of distinct elements: Quality

- For $j \in [n]$ and $r \geq 0$, let $X_{r,j}$ be the indicator r.v. for
  $\text{zeros}(h(j)) \geq r$.
- Since $h(j)$ is uniformly distributed over the log n-bit strings,

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}$$

- Let $Y_r = \sum_{j|f_j>0} X_{r,j}$ and let t denote the final value of z.
- $Y_r > 0$ iff $t \geq r$, or equivalently $Y_r = 0$ iff $t \leq r - 1$.

Counting the number of distinct elements: Quality

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}.$$

Counting the number of distinct elements: Quality

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}.$$

$$E[Y_r] = \sum_{j|f_j>0} E[X_{r,j}] = \frac{d}{2^r}$$

Counting the number of distinct elements: Quality

$$E[X_{r,j}] = Pr[\text{zeros}(h(j)) \geq r] = Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}.$$

$$E[Y_r] = \sum_{j|f_j>0} E[X_{r,j}] = \frac{d}{2^r}$$

- Random variables Y_r are pairwise independent, as they come from a universal hash family.

$$Var[Y_r] = \sum_{j|f_j>0} Var[X_{r,j}] \leq \sum_{j|f_j>0} E[X_{r,j}^2] = \sum_{j|f_j>0} E[X_{r,j}] = \frac{d}{2^r}$$

Counting the number of distinct elements: Quality

- $E[Y_r] = Var[Y_r] = d/2^r$
- Using Markov's and Chebyshev's inequalities,

$$Pr[Y_r > 0] = Pr[Y_r \geq 1] \leq \frac{E[Y_r]}{1} = \frac{d}{2^r}.$$

$$Pr[Y_r = 0] = Pr[|Y_r - E[Y_r]| \geq \frac{d}{2^r}] \leq \frac{Var[Y_r]}{(d/2^r)^2} \leq \frac{2^r}{d}.$$

Counting the number of distinct elements: Quality

- $Pr[Y_r > 0] \leq \frac{d}{2^r}$ and $Pr[Y_r = 0] \leq \frac{2^r}{d}$.

Counting the number of distinct elements: Quality

- $Pr[Y_r > 0] \leq \frac{d}{2^r}$ and $Pr[Y_r = 0] \leq \frac{2^r}{d}$.
- Let $\hat{d}$ be the estimate of d, $\hat{d} = 2^{t+\frac{1}{2}}$.

Counting the number of distinct elements: Quality

- $Pr[Y_r > 0] \leq \frac{d}{2^r}$ and $Pr[Y_r = 0] \leq \frac{2^r}{d}$.
- Let $\hat{d}$ be the estimate of d, $\hat{d} = 2^{t+\frac{1}{2}}$.
- Let a be the smallest integer so that $2^{a+\frac{1}{2}} \geq 3d$,

$$Pr[\hat{d} \geq 3d] = Pr[t \geq a] = Pr[Y_a = 0] \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}.$$

Counting the number of distinct elements: Quality

- $Pr[Y_r > 0] \leq \frac{d}{2^r}$ and $Pr[Y_r = 0] \leq \frac{2^r}{d}$.
- Let $\hat{d}$ be the estimate of d, $\hat{d} = 2^{t+\frac{1}{2}}$.
- Let a be the smallest integer so that $2^{a+\frac{1}{2}} \geq 3d$,

$$Pr[\hat{d} \geq 3d] = Pr[t \geq a] = Pr[Y_a = 0] \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}.$$

- Let b be the largest integer so that $2^{b+\frac{1}{2}} \leq 3d$,

$$Pr[\hat{d} \leq 3d] = Pr[t \leq b] = Pr[Y_{b+1} = 0] \leq \frac{2^{b+1}}{d} \leq \frac{\sqrt{2}}{3}.$$

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k independent copies of the algorithm and take the best information from them, in this case,

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k independent copies of the algorithm and take the best information from them, in this case, the median of the k answers.

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.

- How to improve the quality of the approximation?
- Usual technique: run k independent copies of the algorithm and take the best information from them, in this case, the median of the k answers.
  If the median exceed $3d$ at least $k/2$ of the runs do.

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k independent copies of the algorithm and take the best information from them, in this case, the median of the k answers.
  If the median exceed $3d$ at least $k/2$ of the runs do.
- By standard Chernoff bounds, the median exceed $3d$ with probability $2^{-\Omega(k)}$ and the median is below $3d$ with probability $2^{-\Omega(k)}$.

Counting the number of distinct elements: Quality

- $Pr[\hat{d} \geq 3d] \leq \frac{\sqrt{2}}{3}$ and $Pr[\hat{d} \leq 3d] \leq \frac{\sqrt{2}}{3}$.
- Thus the algorithm provides a $(2, \frac{\sqrt{2}}{3})$-approximation.
- How to improve the quality of the approximation?
- Usual technique: run k independent copies of the algorithm and take the best information from them, in this case, the median of the k answers.
  If the median exceed $3d$ at least $k/2$ of the runs do.
- By standard Chernoff bounds, the median exceed $3d$ with probability $2^{-\Omega(k)}$ and the median is below $3d$ with probability $2^{-\Omega(k)}$.
- Choosing $k = \Theta(\log(1/\delta))$, we can make the sum to be at most δ. So we get a $(2, \delta)$-approximation. However, the used memory is now $O(log(1/\delta) \log n)$.