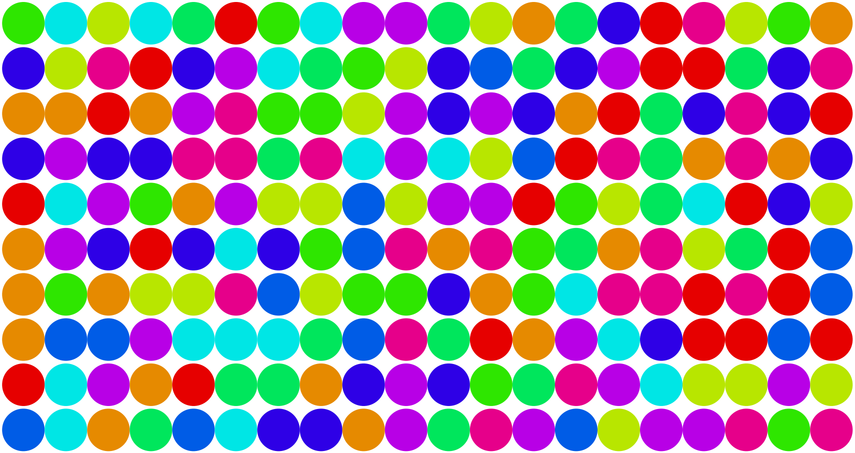
Topics in TCS

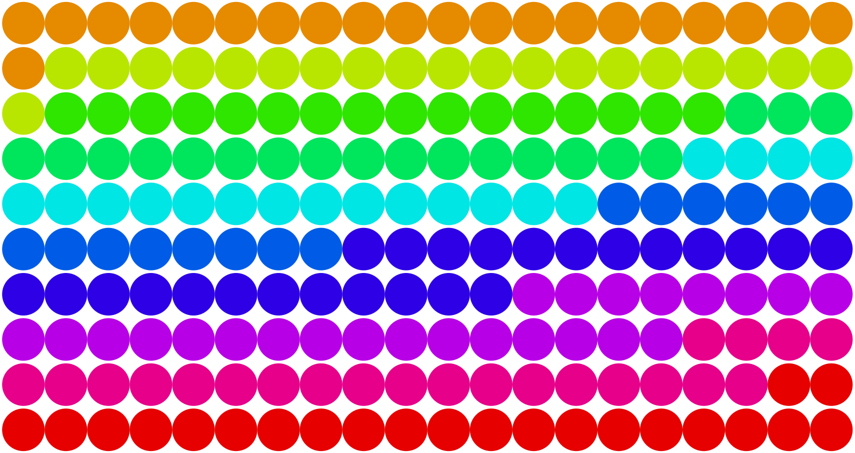
Counting distinct elements

Raphaël Clifford

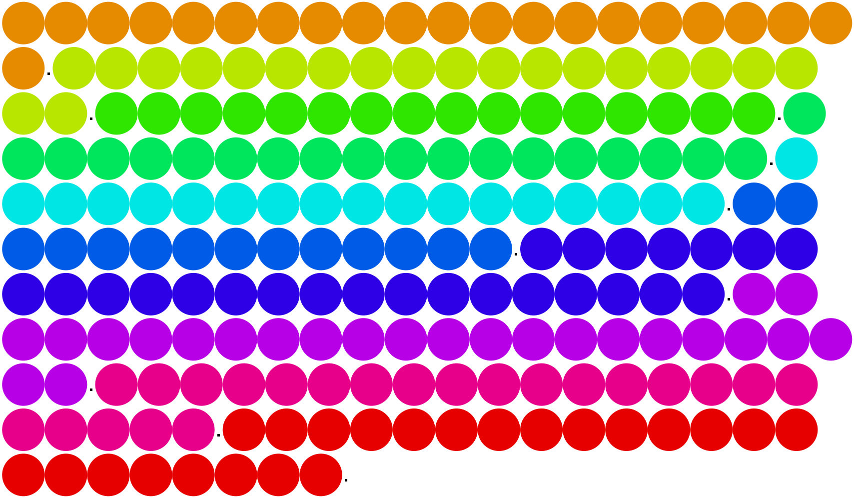
Counting distinct elements



Counting distinct elements



Counting distinct elements



10 distinct colours

Counting distinct elements

Naive counting solution

Sort the elements. $O(m \log m)$ time.

Counting distinct elements

Naive counting solution

Sort the elements. $O(m \log m)$ time.

Traverse from left to right. $O(m)$ time.

Counting distinct elements

Naive counting solution

Sort the elements. $O(m \log m)$ time.

Traverse from left to right. $O(m)$ time.

$O(m \log m)$ time overall BUT $O(m)$ words of space.

Counting distinct elements

Naive counting solution

Sort the elements. $O(m \log m)$ time.

Traverse from left to right. $O(m)$ time.

$O(m \log m)$ time overall BUT $O(m)$ words of space.

NOT one-pass.

Counting distinct elements

Slightly less naive counting solution

Use a balanced binary search tree.

Counting distinct elements

Slightly less naive counting solution

Use a balanced binary search tree.

Traverse values from left to right

Counting distinct elements

Slightly less naive counting solution

Use a balanced binary search tree.

Traverse values from left to right

- If value not in tree, insert it

Counting distinct elements

Slightly less naive counting solution

Use a balanced binary search tree.

Traverse values from left to right

- If value not in tree, insert it

At the conclusion the number of distinct values will be the number of nodes in tree.

Counting distinct elements

Slightly less naive counting solution

Use a balanced binary search tree.

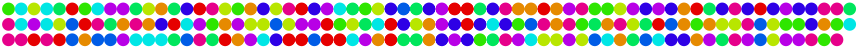
Traverse values from left to right

- If value not in tree, insert it

At the conclusion the number of distinct values will be the number of nodes in tree.

Running time $O(\log m)$ per FIND and INSERT operation making $O(m \log m)$ time overall BUT $O(m)$ words of space.

Counting distinct elements



Slightly less naive counting solution

Use a balanced binary search tree.

Traverse values from left to right

- If value not in tree, insert it

At the conclusion the number of distinct values will be the number of nodes in tree.

Running time $O(\log m)$ per Find and Insert operation making $O(m \log m)$ time overall BUT $O(m)$ words of space.

One-pass ✓

Counting distinct elements

Slightly less naive counting solution

Use a balanced binary search tree.

Traverse values from left to right

- If value not in tree, insert it

At the conclusion the number of distinct values will be the number of nodes in tree.

Running time $O(\log m)$ per FIND and INSERT operation making $O(m \log m)$ time overall BUT $O(m)$ words of space.

One-pass ✓

No deterministic sub-linear space one-pass solution is possible.

Counting distinct elements

Slightly less naive counting solution

Use a balanced binary search tree.

Traverse values from left to right

If value not in tree, insert it

At the conclusion the number of distinct values will be the number of nodes in tree.

Running time $O(\log m)$ per Find and Insert operation making $O(m \log m)$ time overall BUT $O(m)$ words of space.

One-pass ✓

No deterministic sub-linear space one-pass solution is possible.

We will need a randomised algorithm.

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

Worked example

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

Worked example

- Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$.
  $n = 10$.

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

Simple-Count
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

Worked example

- Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.
- Randomly hash to $\langle 10, 9, 10, 6, 9, 10, 6, 9 \rangle$

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

Worked example

- Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.
- Randomly hash to $\langle 10, 9, 10, 6, 9, 10, 6, 9 \rangle$
- $M = 6$ at termination.

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

Worked example

- Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.
- Randomly hash to $\langle 10, 9, 10, 6, 9, 10, 6, 9 \rangle$
- $M = 6$ at termination.
- Return $\hat{d} = 10/6 = 1\frac{2}{3}$

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

Worked example

- Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.
- Randomly hash to $\langle 10, 9, 10, 6, 9, 10, 6, 9 \rangle$
- $M = 6$ at termination.
- Return $\hat{d} = 10/6 = 1\frac{2}{3}$
- True answer is 3.

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

Worked example. 2nd try

- Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.
- Rehash: $\langle 4, 3, 4, 8, 3, 4, 8, 3 \rangle$
-
-
- True answer is 3.

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
        set M = h(a_i)

return d̂ = n/M
```

Worked example. 2nd try

- Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.
- Rehash: $\langle 4, 3, 4, 8, 3, 4, 8, 3 \rangle$
- $M = 3$ at termination.
-
- True answer is 3.

First randomised algorithm

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
        set M = h(a_i)

return d̂ = n/M
```

Worked example. 2nd try

- Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.
- Rehash: $\langle 4, 3, 4, 8, 3, 4, 8, 3 \rangle$
- $M = 3$ at termination.
- Return $\hat{d} = 10/3 = 3\frac{1}{3}$
- True answer is 3.

Probability lemma for counting distinct elements

Lemma

Let $(Z_1, \ldots, Z_N)$ *be an array of pairwise independent indicator variables with* $\Pr(Z_i = 1) = p$ *and let* $W = \sum_{i=1}^{N} Z_i$*, then* $\mathbb{E}(W) = Np$, $\text{var}(W) = Np(1-p)$ *and*

$$\Pr(W > 0) \leq Np \quad \textit{and} \quad \Pr(W = 0) \leq \frac{1}{Np} \tag{1}$$

Proof.

□

Probability lemma for counting distinct elements

Lemma

Let $(Z_1, \ldots, Z_N)$ *be an array of pairwise independent indicator variables with* $\Pr(Z_i = 1) = p$ *and let* $W = \sum_{i=1}^{N} Z_i$*, then* $\mathbb{E}(W) = Np$*,* $\text{var}(W) = Np(1-p)$ *and*

$$\Pr(W > 0) \leq Np \quad \textit{and} \quad \Pr(W = 0) \leq \frac{1}{Np} \tag{1}$$

Proof.

- $\mathbb{E}(W) = Np$, $\text{var}(W) = Np(1-p)$ both follow from linearity of expectation and the preliminary probability notes.

□

Probability lemma for counting distinct elements

Lemma

Let $(Z_1, \ldots, Z_N)$ *be an array of pairwise independent indicator variables with* $\Pr(Z_i = 1) = p$ *and let* $W = \sum_{i=1}^{N} Z_i$, *then* $\mathbb{E}(W) = Np$, $\text{var}(W) = Np(1-p)$ *and*

$$\Pr(W > 0) \leq Np \quad \textit{and} \quad \Pr(W = 0) \leq \frac{1}{Np} \tag{1}$$

Proof.

- $\mathbb{E}(W) = Np$, $\text{var}(W) = Np(1-p)$ both follow from linearity of expectation and the preliminary probability notes.
- $\Pr(W > 0) \leq Np$ is by the union bound and that $\Pr(Z_i = 1) = p$.

□

Probability lemma for counting distinct elements

Lemma

Let $(Z_1, \ldots, Z_N)$ *be an array of pairwise independent indicator variables with* $\Pr(Z_i = 1) = p$ *and let* $W = \sum_{i=1}^{N} Z_i$*, then* $\mathbb{E}(W) = Np$, $\text{var}(W) = Np(1-p)$ *and*

$$\Pr(W > 0) \leq Np \quad \textit{and} \quad \Pr(W = 0) \leq \frac{1}{Np} \tag{1}$$

Proof.

- $\mathbb{E}(W) = Np$, $\text{var}(W) = Np(1-p)$ both follow from linearity of expectation and the preliminary probability notes.
- $\Pr(W > 0) \leq Np$ is by the union bound and that $\Pr(Z_i = 1) = p$.
- $\Pr(W = 0) \leq \Pr(|W - \mathbb{E}(W)| \geq \mathbb{E}(W)) \leq \frac{\text{var}(W)}{(Np)^2} \leq \frac{1}{Np}$

□

SIMPLE-COUNT analysis

- Let us consider the set of distinct values $S = \{h(a_i), i \in [m]\}$.

SIMPLE-COUNT analysis

- Let us consider the set of distinct values $S = \{h(a_i), i \in [m]\}$.
- Let r.v. Y_a be the number of elements that are hashed to a value less than or equal to a, for arbitrary a.

SIMPLE-COUNT analysis

- Let us consider the set of distinct values $S = \{h(a_i), i \in [m]\}$.
- Let r.v. Y_a be the number of elements that are hashed to a value less than or equal to a, for arbitrary a.
- Y_a corresponds to W from (1) with $N = d$ and $p = a/n$. Therefore:

$$\Pr(Y_a > 0) = \Pr(M \leq a) \leq da/n \quad \text{and} \quad \Pr(M > b) \leq \frac{n}{db} \quad (2)$$

SIMPLE-COUNT analysis

- Let us consider the set of distinct values $S = \{h(a_i), i \in [m]\}$.
- Let r.v. Y_a be the number of elements that are hashed to a value less than or equal to a, for arbitrary a.
- Y_a corresponds to W from (1) with $N = d$ and $p = a/n$. Therefore:

$$\Pr(Y_a > 0) = \Pr(M \le a) \le da/n \quad \text{and} \quad \Pr(M > b) \le \frac{n}{db} \quad (2)$$

There is an element less than or equal to a iff the min is at most a

Simple-Count analysis

- Let us consider the set of distinct values $S = \{h(a_i), i \in [m]\}$.
- Let r.v. Y_a be the number of elements that are hashed to a value less than or equal to a, for arbitrary a.
- Y_a corresponds to W from (1) with $N = d$ and $p = a/n$. Therefore:

$$\Pr(Y_a > 0) = \Pr(M \leq a) \leq da/n \quad \text{and} \quad \Pr(M > b) \leq \frac{n}{db} \quad (2)$$

There is an element less than or equal to a iff the min is at most a

Define $Z_i = 1$ if $h(S_i) \leq b$ and 0 otherwise
Let the random variable $Z = \sum_{i=1}^{d} Z_i$
From (1), $\Pr(M > b) = \Pr(Z = 0) \leq \frac{1}{\frac{db}{n}} = \frac{n}{db}$

Simple-Count analysis

- Let us consider the set of distinct values $S = \{h(a_i), i \in [m]\}$.
- Let r.v. Y_a be the number of elements that are hashed to a value less than or equal to a, for arbitrary a.
- Y_a corresponds to W from (1) with $N = d$ and $p = a/n$. Therefore:

$$\Pr(Y_a > 0) = \Pr(M \leq a) \leq da/n \quad \text{and} \quad \Pr(M > b) \leq \frac{n}{db} \quad (2)$$

- Now let $da/n = 1/3$ and $n/db = 1/3$, so that $a = n/(3d)$ and $b = 3n/d$, then (2) becomes

$$\Pr(M \leq n/(3d)) \leq 1/3 \quad \text{and} \quad \Pr(M > 3n/d) \leq 1/3$$

or

$$\Pr(d \leq \hat{d}/3) \leq 1/3 \quad \text{and} \quad \Pr(d > 3\hat{d}) \leq 1/3$$

using $\hat{d} = n/M$.

Simple-Count space/time

Our random estimate $\hat{d}$ gives us:

$$\Pr(d \leq \hat{d}/3) \leq 1/3 \quad \text{and} \quad \Pr(d > 3\hat{d}) \leq 1/3$$

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
Choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

Simple-Count space/time

Our random estimate $\hat{d}$ gives us:

$$\Pr(d \leq \hat{d}/3) \leq 1/3 \quad \text{and} \quad \Pr(d > 3\hat{d}) \leq 1/3$$

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
Choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

- Running time $O(m)$

SIMPLE-COUNT space/time

Our random estimate $\hat{d}$ gives us:

$$\Pr(d \leq \hat{d}/3) \leq 1/3 \quad \text{and} \quad \Pr(d > 3\hat{d}) \leq 1/3$$

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
Choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

- Running time $O(m)$
- One-pass ✓

SIMPLE-COUNT space/time

Our random estimate $\hat{d}$ gives us:

$$\Pr(d \leq \hat{d}/3) \leq 1/3 \quad \text{and} \quad \Pr(d > 3\hat{d}) \leq 1/3$$

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
Choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

- Running time $O(m)$
- One-pass ✓
- Space $O(\log n)$ ✓

Simple-Count space/time

Our random estimate $\hat{d}$ gives us:

$$\Pr(d \le \hat{d}/3) \le 1/3 \quad \text{and} \quad \Pr(d > 3\hat{d}) \le 1/3$$

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
Choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

- Running time $O(m)$
- One-pass ✓
- Space $O(\log n)$ ✓
- Can we use less space? (Sort of)

Simple-Count space/time

Our random estimate $\hat{d}$ gives us:

$$\Pr(d \leq \hat{d}/3) \leq 1/3 \quad \text{and} \quad \Pr(d > 3\hat{d}) \leq 1/3$$

```
stream ⟨a_1, ..., a_m⟩, a_i ∈ [n]
initialise
Choose random h : [n] → [n]

SIMPLE-COUNT
Set M = h(a_1)
For each i ≥ 2
   if h(a_i) < M
         set M = h(a_i)

return d̂ = n/M
```

- Running time $O(m)$
- One-pass ✓
- Space $O(\log n)$ ✓
- Can we use less space? (Sort of)
- The error probability is pretty bad. Can we fix that? (Yes)

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^{z+1/2}
```

- h chosen from a pairwise random family of hash functions.

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^{z+1/2}
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

Let's try it. Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

Let's try it. Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.

- Hashed to $\langle 8, 2, 8, 1, 2, 8, 1, 2 \rangle$

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^{z+1/2}
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

Let's try it. Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.

- Hashed to $\langle 8, 2, 8, 1, 2, 8, 1, 2 \rangle$
- In binary: $\langle 1000, 10, 1000, 1, 10, 1000, 1, 10 \rangle$

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

Let's try it. Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.

- Hashed to $\langle 8, 2, 8, 1, 2, 8, 1, 2 \rangle$
- In binary: $\langle 1000, 10, 1000, 1, 10, 1000, 1, 10 \rangle$
- Max value of ZEROS is 3. Return $2^{3+1/2} \approx 11.3$

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

Let's try it. Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.

- Hashed to $\langle 8, 2, 8, 1, 2, 8, 1, 2 \rangle$
- In binary: $\langle 1000, 10, 1000, 1, 10, 1000, 1, 10 \rangle$
- Max value of ZEROS is 3. Return $2^{3+1/2} \approx 11.3$
- True value 3. What happens if we pick another hash function?

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

Let's try it. Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.

- Hashed to $\langle 4, 3, 4, 1, 3, 4, 1, 3 \rangle$

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^{z+1/2}
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

Let's try it. Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.

- Hashed to $\langle 4, 3, 4, 1, 3, 4, 1, 3 \rangle$
- In binary: $\langle 100, 11, 100, 1, 11, 100, 1, 11 \rangle$

The TIDEMARK algorithm

```
initialise
choose random h : [n] → [n]
set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- h chosen from a pairwise random family of hash functions.
- ZEROS counts the number of trailing zeros in the binary representation of a positive integer.
- Finds the maximum value of ZEROS$(h(a_i))$ overall.

Let's try it. Stream $\langle 5, 4, 5, 7, 4, 5, 7, 4 \rangle$. $n = 10$.

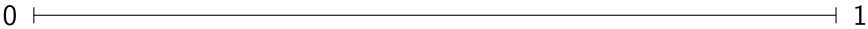
- Hashed to $\langle 4, 3, 4, 1, 3, 4, 1, 3 \rangle$
- In binary: $\langle 100, 11, 100, 1, 11, 100, 1, 11 \rangle$
- Max value of ZEROS is 2. Return $2^{2+1/2} \approx 5.7$. We were luckier.

Understanding the TIDEMARK algorithm

$$\textsc{Zeros}(x) = 0 \qquad x = 1, 3, 5, \ldots$$
$$\textsc{Zeros}(x) \geq 1 \qquad x = 2, 4, 6, \ldots$$
$$\textsc{Zeros}(x) \geq 2 \qquad x = 4, 8, 12, \ldots$$
$$\textsc{Zeros}(x) \geq 3 \qquad x = 8, 16, 24, \ldots$$
$$\textsc{Zeros}(x) \geq 4 \qquad x = 16, 32, 48, \ldots$$

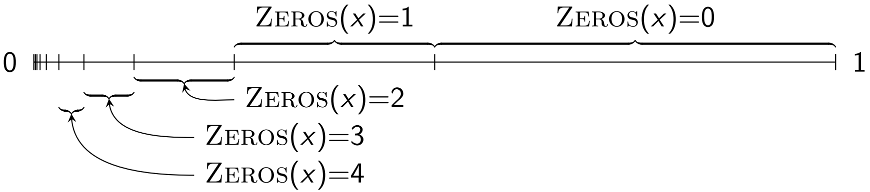
Understanding the TIDEMARK algorithm

$\text{ZEROS}(x) = 0$	$x = 1, 3, 5, \ldots$	$p = 1/2$
$\text{ZEROS}(x) \geq 1$	$x = 2, 4, 6, \ldots$	$p = 1/2$
$\text{ZEROS}(x) \geq 2$	$x = 4, 8, 12, \ldots$	$p = 1/4$
$\text{ZEROS}(x) \geq 3$	$x = 8, 16, 24, \ldots$	$p = 1/8$
$\text{ZEROS}(x) \geq 4$	$x = 16, 32, 48, \ldots$	$p = 1/16$



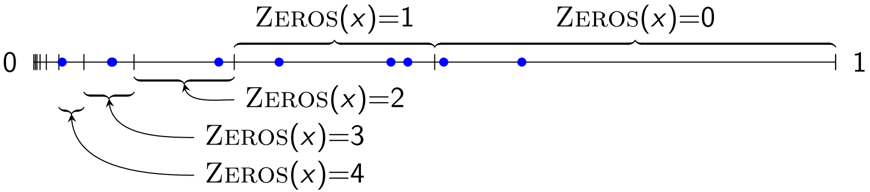
Understanding the Tidemark algorithm

$\textsc{Zeros}(x) = 0$	$x = 1, 3, 5, \ldots$	$p = 1/2$
$\textsc{Zeros}(x) \geq 1$	$x = 2, 4, 6, \ldots$	$p = 1/2$
$\textsc{Zeros}(x) \geq 2$	$x = 4, 8, 12, \ldots$	$p = 1/4$
$\textsc{Zeros}(x) \geq 3$	$x = 8, 16, 24, \ldots$	$p = 1/8$
$\textsc{Zeros}(x) \geq 4$	$x = 16, 32, 48, \ldots$	$p = 1/16$



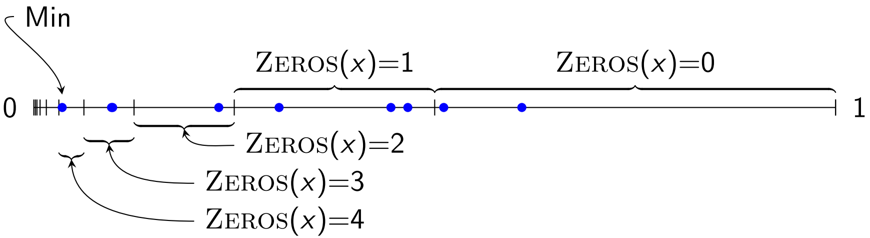
Understanding the TIDEMARK algorithm

$\textsc{Zeros}(x) = 0$	$x = 1, 3, 5, \ldots$	$p = 1/2$
$\textsc{Zeros}(x) \geq 1$	$x = 2, 4, 6, \ldots$	$p = 1/2$
$\textsc{Zeros}(x) \geq 2$	$x = 4, 8, 12, \ldots$	$p = 1/4$
$\textsc{Zeros}(x) \geq 3$	$x = 8, 16, 24, \ldots$	$p = 1/8$
$\textsc{Zeros}(x) \geq 4$	$x = 16, 32, 48, \ldots$	$p = 1/16$



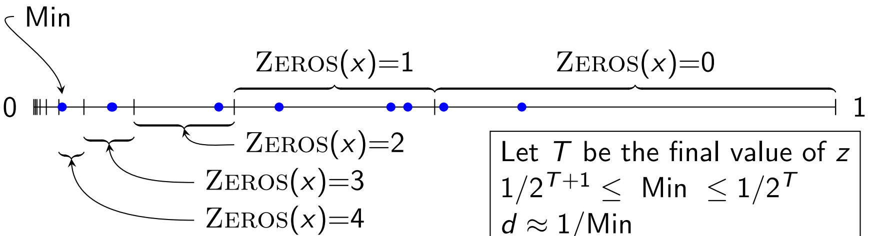
Understanding the TIDEMARK algorithm

$\textsc{Zeros}(x) = 0$	$x = 1, 3, 5, \ldots$	$p = 1/2$
$\textsc{Zeros}(x) \geq 1$	$x = 2, 4, 6, \ldots$	$p = 1/2$
$\textsc{Zeros}(x) \geq 2$	$x = 4, 8, 12, \ldots$	$p = 1/4$
$\textsc{Zeros}(x) \geq 3$	$x = 8, 16, 24, \ldots$	$p = 1/8$
$\textsc{Zeros}(x) \geq 4$	$x = 16, 32, 48, \ldots$	$p = 1/16$



Understanding the TIDEMARK algorithm

$\text{ZEROS}(x) = 0$	$x = 1, 3, 5, \ldots$	$p = 1/2$
$\text{ZEROS}(x) \geq 1$	$x = 2, 4, 6, \ldots$	$p = 1/2$
$\text{ZEROS}(x) \geq 2$	$x = 4, 8, 12, \ldots$	$p = 1/4$
$\text{ZEROS}(x) \geq 3$	$x = 8, 16, 24, \ldots$	$p = 1/8$
$\text{ZEROS}(x) \geq 4$	$x = 16, 32, 48, \ldots$	$p = 1/16$



Let T be the final value of z
$1/2^{T+1} \leq$ Min $\leq 1/2^{T}$
$d \approx 1/\text{Min}$
$\implies d \approx 2^{T+1/2}$

The tidemark algorithm - analysis I

For $j \in [n]$, define indicator r.v. $X_{r,j} = 1$ if $\textsc{zeros}(h(j)) \geq r$.

The tidemark algorithm - analysis I

For $j \in [n]$, define indicator r.v. $X_{r,j} = 1$ if $\textsc{zeros}(h(j)) \geq r$.

Define r.v. $Y_r = \sum_{j, f_j > 0} X_{r,j}$. That is the number of hashed values in the stream that have at least r trailing zeros.

The tidemark algorithm - analysis I

For $j \in [n]$, define indicator r.v. $X_{r,j} = 1$ if $\textsc{zeros}(h(j)) \geq r$.

Define r.v. $Y_r = \sum_{j, f_j > 0} X_{r,j}$. That is the number of hashed values in the stream that have at least r trailing zeros.

Let T be the final value of z,

The tidemark algorithm - analysis I

For $j \in [n]$, define indicator r.v. $X_{r,j} = 1$ if $\textsc{zeros}(h(j)) \geq r$.

Define r.v. $Y_r = \sum_{j, f_j > 0} X_{r,j}$. That is the number of hashed values in the stream that have at least r trailing zeros.

Let T be the final value of z,

$$Y_r > 0 \text{ iff } T \geq r$$

The tidemark algorithm - analysis I

For $j \in [n]$, define indicator r.v. $X_{r,j} = 1$ if $\textsc{zeros}(h(j)) \geq r$.

Define r.v. $Y_r = \sum_{j, f_j > 0} X_{r,j}$. That is the number of hashed values in the stream that have at least r trailing zeros.

Let T be the final value of z,

$$Y_r > 0 \text{ iff } T \geq r$$
$$Y_r = 0 \text{ iff } T \leq r - 1$$

The tidemark algorithm - analysis I

For $j \in [n]$, define indicator r.v. $X_{r,j} = 1$ if $\textsc{zeros}(h(j)) \geq r$.

Define r.v. $Y_r = \sum_{j, f_j > 0} X_{r,j}$. That is the number of hashed values in the stream that have at least r trailing zeros.

Let T be the final value of z,

$$Y_r > 0 \text{ iff } T \geq r$$

$$Y_r = 0 \text{ iff } T \leq r - 1$$

$$\mathbb{E}(X_{r,j}) = \Pr(\textsc{zeros}(h(j)) \geq r) = \Pr(2^r \text{ divides } h(j))$$

The tidemark algorithm - analysis I

For $j \in [n]$, define indicator r.v. $X_{r,j} = 1$ if $\textsc{zeros}(h(j)) \geq r$.

Define r.v. $Y_r = \sum_{j, f_j > 0} X_{r,j}$. That is the number of hashed values in the stream that have at least r trailing zeros.

Let T be the final value of z,

$$\begin{aligned} Y_r > 0 &\text{ iff } T \geq r \\ Y_r = 0 &\text{ iff } T \leq r - 1 \\ \mathbb{E}(X_{r,j}) = \Pr(\textsc{zeros}(h(j)) \geq r) &= \Pr(2^r \text{ divides } h(j)) \end{aligned}$$

Assuming $2^r \leq n$ then

$$1/2^r \leq \Pr(2^r \text{ divides } h(j)) \leq 1/2^{r-1}$$

The tidemark algorithm - analysis I

For $j \in [n]$, define indicator r.v. $X_{r,j} = 1$ if $\textsc{zeros}(h(j)) \geq r$.

Define r.v. $Y_r = \sum_{j, f_j > 0} X_{r,j}$. That is the number of hashed values in the stream that have at least r trailing zeros.

Let T be the final value of z,

$$\begin{aligned} Y_r > 0 &\text{ iff } T \geq r \\ Y_r = 0 &\text{ iff } T \leq r - 1 \\ \mathbb{E}(X_{r,j}) &= \Pr(\textsc{zeros}(h(j)) \geq r) = \Pr(2^r \text{ divides } h(j)) \end{aligned}$$

Assuming $2^r \leq n$ then

$$1/2^r \leq \Pr(2^r \text{ divides } h(j)) \leq 1/2^{r-1}$$

If n is a power of 2, $\mathbb{E}(X_{r,j}) = 1/2^r$

The tidemark algorithm - analysis II

For simplicity, assume n is a power of 2.

$$\mathbb{E}(Y_r) = \sum_{j: f_j > 0} \mathbb{E}(X_{r,j}) \quad = \frac{d}{2^r}$$

The tidemark algorithm - analysis II

For simplicity, assume n is a power of 2.

$$\mathbb{E}(Y_r) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) \quad = \frac{d}{2^r}$$

$$\text{var}(Y_r) = \sum_{j:f_j>0} \text{var}(X_{r,j}) \leq \sum_{j:f_j>0} \mathbb{E}(X_{r,j}^2) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) = \frac{d}{2^r}$$

The tidemark algorithm - analysis II

For simplicity, assume n is a power of 2.

$$\mathbb{E}(Y_r) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) \quad = \frac{d}{2^r}$$

$$\mathrm{var}(Y_r) = \sum_{j:f_j>0} \mathrm{var}(X_{r,j}) \leq \sum_{j:f_j>0} \mathbb{E}(X_{r,j}^2) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) = \frac{d}{2^r}$$

By Markov's inequality,

$$\Pr(Y_r > 0) = \Pr(Y_r \geq 1) \leq \frac{\mathbb{E}(Y_r)}{1} = \frac{d}{2^r}$$

The tidemark algorithm - analysis II

For simplicity, assume n is a power of 2.

$$\mathbb{E}(Y_r) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) \quad = \frac{d}{2^r}$$

$$\mathrm{var}(Y_r) = \sum_{j:f_j>0} \mathrm{var}(X_{r,j}) \leq \sum_{j:f_j>0} \mathbb{E}(X_{r,j}^2) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) = \frac{d}{2^r}$$

By Markov's inequality,

$$\Pr(Y_r > 0) = \Pr(Y_r \geq 1) \leq \frac{\mathbb{E}(Y_r)}{1} = \frac{d}{2^r}$$

By Chebyshev's inequality,

$$\Pr(Y_r = 0) \leq \Pr(|Y_r - \mathbb{E}(Y_r)| \geq d/2^r) \leq \frac{\mathrm{var}(Y_r)}{(d/2^r)^2} \leq \frac{2^r}{d}$$

The tidemark algorithm - analysis II

For simplicity, assume n is a power of 2.

$$\mathbb{E}(Y_r) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) \quad = \frac{d}{2^r}$$

$$\mathrm{var}(Y_r) = \sum_{j:f_j>0} \mathrm{var}(X_{r,j}) \leq \sum_{j:f_j>0} \mathbb{E}(X_{r,j}^2) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) = \frac{d}{2^r}$$

By Markov's inequality,

$$\Pr(Y_r > 0) = \Pr(Y_r \geq 1) \leq \frac{\mathbb{E}(Y_r)}{1} = \frac{d}{2^r}$$

By Chebyshev's inequality,

$$\Pr(Y_r = 0) \leq \Pr(|Y_r - \mathbb{E}(Y_r)| \geq d/2^r) \leq \frac{\mathrm{var}(Y_r)}{(d/2^r)^2} \leq \frac{2^r}{d}$$

The tidemark algorithm - analysis II

For simplicity, assume n is a power of 2.

$$\mathbb{E}(Y_r) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) \quad = \frac{d}{2^r}$$

$$\text{var}(Y_r) = \sum_{j:f_j>0} \text{var}(X_{r,j}) \leq \sum_{j:f_j>0} \mathbb{E}(X_{r,j}^2) = \sum_{j:f_j>0} \mathbb{E}(X_{r,j}) = \frac{d}{2^r}$$

By Markov's inequality,

$$\Pr(Y_r > 0) = \Pr(Y_r \geq 1) \leq \frac{\mathbb{E}(Y_r)}{1} = \frac{d}{2^r}$$

By Chebyshev's inequality,

$$\Pr(Y_r = 0) \leq \Pr(|Y_r - \mathbb{E}(Y_r)| \geq d/2^r) \leq \frac{\text{var}(Y_r)}{(d/2^r)^2} \leq \frac{2^r}{d}$$

The tidemark algorithm - analysis III

Recall: $\hat{d} = 2^{T+1/2}$, $\Pr(Y_r > 0) \leq \frac{d}{2^r}$ and $\Pr(Y_r = 0) \leq \frac{2^r}{d}$.

The tidemark algorithm - analysis III

Recall: $\hat{d} = 2^{T+1/2}$, $\Pr(Y_r > 0) \leq \frac{d}{2^r}$ and $\Pr(Y_r = 0) \leq \frac{2^r}{d}$.

- Let a be the smallest integer such that $2^{a+1/2} \geq 3d$. We have,

$$\Pr(\hat{d} \geq 3d) = \Pr(T \geq a)$$

The tidemark algorithm - analysis III

Recall: $\hat{d} = 2^{T+1/2}$, $\Pr(Y_r > 0) \le \frac{d}{2^r}$ and $\Pr(Y_r = 0) \le \frac{2^r}{d}$.

- Let a be the smallest integer such that $2^{a+1/2} \ge 3d$. We have,

$$\Pr(\hat{d} \ge 3d) = \Pr(T \ge a) = \Pr(Y_a > 0) \le \frac{d}{2^a} \le \frac{\sqrt{2}}{3}$$

The tidemark algorithm - analysis III

Recall: $\hat{d} = 2^{T+1/2}$, $\Pr(Y_r > 0) \leq \frac{d}{2^r}$ and $\Pr(Y_r = 0) \leq \frac{2^r}{d}$.

- Let a be the smallest integer such that $2^{a+1/2} \geq 3d$. We have,

$$\Pr(\hat{d} \geq 3d) = \Pr(T \geq a) = \Pr(Y_a > 0) \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}$$

$$2^{a+1/2} \geq 3d \implies d \leq \frac{\sqrt{2} \cdot 2^a}{3} \implies \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}$$

The tidemark algorithm - analysis III

Recall: $\hat{d} = 2^{T+1/2}$, $\Pr(Y_r > 0) \leq \frac{d}{2^r}$ and $\Pr(Y_r = 0) \leq \frac{2^r}{d}$.

- Let a be the smallest integer such that $2^{a+1/2} \geq 3d$. We have,

$$\Pr(\hat{d} \geq 3d) = \Pr(T \geq a) = \Pr(Y_a > 0) \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}$$

  This gives us the probability that our estimate is too large. We now bound the probability it is too small.

The tidemark algorithm - analysis III

Recall: $\hat{d} = 2^{T+1/2}$, $\Pr(Y_r > 0) \leq \frac{d}{2^r}$ and $\Pr(Y_r = 0) \leq \frac{2^r}{d}$.

- Let a be the smallest integer such that $2^{a+1/2} \geq 3d$. We have,

$$\Pr(\hat{d} \geq 3d) = \Pr(T \geq a) = \Pr(Y_a > 0) \leq \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}$$

  This gives us the probability that our estimate is too large. We now bound the probability it is too small.

- For the probability that our estimate is too small let b be the largest integer such that $2^{b+1/2} \leq d/3$.

$$\Pr(\hat{d} \leq d/3) = \Pr(T \leq b) = \Pr(Y_{b+1} = 0) \leq \frac{2^{b+1}}{d} \leq \frac{\sqrt{2}}{3}$$

The tidemark algorithm - analysis III

Recall: $\hat{d} = 2^{T+1/2}$, $\Pr(Y_r > 0) \le \frac{d}{2^r}$ and $\Pr(Y_r = 0) \le \frac{2^r}{d}$.

- Let a be the smallest integer such that $2^{a+1/2} \ge 3d$. We have,

$$\Pr(\hat{d} \ge 3d) = \Pr(T \ge a) = \Pr(Y_a > 0) \le \frac{d}{2^a} \le \frac{\sqrt{2}}{3}$$

  This gives us the probability that our estimate is too large. We now bound the probability it is too small.

- For the probability that our estimate is too small let b be the largest integer such that $2^{b+1/2} \le d/3$.

$$\Pr(\hat{d} \le d/3) = \Pr(T \le b) = \Pr(Y_{b+1} = 0) \le \frac{2^{b+1}}{d} \le \frac{\sqrt{2}}{3}$$

The bounds are not ideal in two ways.

1. We can't get an arbitrarily close approximation (yet).
2. The failure probably is high. $\sqrt{2}/3 \approx 0.47$!

TIDEMARK space/time

```
initialise
Choose random h : [n] → [n]
Set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

TIDEMARK space/time

```
initialise
Choose random h : [n] → [n]
Set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^{z+1/2}
```

- One-pass and $O(m)$ time. ✓

TIDEMARK space/time

```
initialise
Choose random h : [n] → [n]
Set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^{z+1/2}
```

- One-pass and $O(m)$ time. ✓
- $O(\log\log n)$ bits of space. ✓

Tidemark space/time

```
initialise
Choose random h : [n] → [n]
Set z = 0

Tidemark(a_i)
if Zeros(h(a_i)) > z
   set z = Zeros(h(a_i))

Output 2^{z+1/2}
```

- One-pass and $O(m)$ time. ✓
- $O(\log \log n)$ bits of space. ✓

Why is it $O(\log \log n)$ bits of space?

TIDEMARK space/time

```
initialise
Choose random h : [n] → [n]
Set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- One-pass and $O(m)$ time. ✓
- $O(\log \log n)$ bits of space. ✓

Why is it $O(\log \log n)$ bits of space?

- We store one value of $\text{ZEROS}(h(a_i))$ where $h(a_i) \in [n]$.

TIDEMARK space/time

```
initialise
Choose random h : [n] → [n]
Set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^{z+1/2}
```

- One-pass and $O(m)$ time. ✓
- $O(\log\log n)$ bits of space. ✓

Why is it $O(\log\log n)$ bits of space?

- We store one value of $\textsc{Zeros}(h(a_i))$ where $h(a_i) \in [n]$.
- The maximum value of $\textsc{Zeros}(h(a_i))$ is $\lfloor \log_2 n \rfloor \in O(\log n)$.

TIDEMARK space/time

```
initialise
Choose random h : [n] → [n]
Set z = 0

TIDEMARK(a_i)
if ZEROS(h(a_i)) > z
   set z = ZEROS(h(a_i))

OUTPUT 2^(z+1/2)
```

- One-pass and $O(m)$ time. ✓
- $O(\log \log n)$ bits of space. ✓

Why is it $O(\log \log n)$ bits of space?

- We store one value of $\text{ZEROS}(h(a_i))$ where $h(a_i) \in [n]$.
- The maximum value of $\text{ZEROS}(h(a_i))$ is $\lfloor \log_2 n \rfloor \in O(\log n)$.
- Therefore only $O(\log \log n)$ bits are needed to represent the biggest possible value.

The median trick - upper bound

Method: For each value in the input, run k independent copies of TIDEMARK in parallel and output the median.

The median trick - upper bound

Method: For each value in the input, run k independent copies of TIDEMARK in parallel and output the median.

Recall: $\Pr(\hat{d} \geq 3d) \leq \frac{\sqrt{2}}{3}$

The median trick - upper bound

Method: For each value in the input, run k independent copies of TIDEMARK in parallel and output the median.

Recall: $\Pr(\hat{d} \geq 3d) \leq \frac{\sqrt{2}}{3}$

- If the median is greater than $3d$, then at least $k/2$ values are at least $3d$.

The median trick - upper bound

Method: For each value in the input, run k independent copies of TIDEMARK in parallel and output the median.

Recall: $\Pr(\hat{d} \geq 3d) \leq \frac{\sqrt{2}}{3}$

- If the median is greater than $3d$, then at least $k/2$ values are at least $3d$.

Define $X_i = 1$ if $\hat{d} \geq 3d$, 0 otherwise. $X = \sum_{i=1}^{k} X_i$, $\mu \leq \sqrt{2}k/3$.

Let $\delta = 3/(2\sqrt{2}) - 1 \approx 0.06$ so $(1+\delta)\mu = k/2$.

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left(-\delta^2\mu/3\right) \qquad \text{(Chernoff bound)}$$

The median trick - upper bound

Method: For each value in the input, run k independent copies of TIDEMARK in parallel and output the median.

Recall: $\Pr(\hat{d} \geq 3d) \leq \frac{\sqrt{2}}{3}$

- If the median is greater than $3d$, then at least $k/2$ values are at least $3d$.

Define $X_i = 1$ if $\hat{d} \geq 3d$, 0 otherwise. $X = \sum_{i=1}^{k} X_i$, $\mu \leq \sqrt{2}k/3$.

Let $\delta = 3/(2\sqrt{2}) - 1 \approx 0.06$ so $(1+\delta)\mu = k/2$.

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left(-\delta^2\mu/3\right) \qquad \text{(Chernoff bound)}$$

In other words, as k grows the probability that the median is at least $3d$ decreases exponentially.

The median trick - lower bound

Method: For each value in the input, run k independent copies of TIDEMARK in parallel and output the median.

Recall: $\Pr(\hat{d} \leq d/3) \leq \frac{\sqrt{2}}{3}$

- If the median is less than d/3, then at least $k/2$ values are at most d/3.

Define $X_i = 1$ if $\hat{d} \leq d/3$, 0 otherwise, $X = \sum_{i=1}^{k} X_i$, $\mu \leq \sqrt{2}k/3$.

Let $\delta = 3/(2\sqrt{2}) - 1 \approx 0.06$ so $(1+\delta)\mu = k/2$.

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left(-\delta^2\mu/3\right) \qquad \text{(Chernoff bound)}$$

In other words, as k grows the probability that the median is at most $d/3$ decreases exponentially.

Summary

We saw two streaming algorithms for estimating the number of distinct items in a stream.

They are both one-pass algorithms and run in $O(m)$ time.

They both hash the incoming elements to $[n]$.

Summary

We saw two streaming algorithms for estimating the number of distinct items in a stream.

They are both one-pass algorithms and run in $O(m)$ time.

They both hash the incoming elements to $[n]$.

- SIMPLE-COUNT stores the minimum of the hashed values.

Summary

We saw two streaming algorithms for estimating the number of distinct items in a stream.

They are both one-pass algorithms and run in $O(m)$ time.

They both hash the incoming elements to $[n]$.

- SIMPLE-COUNT stores the minimum of the hashed values.
- SIMPLE-COUNT uses $O(\log n)$ bits of space.

Summary

We saw two streaming algorithms for estimating the number of distinct items in a stream.

They are both one-pass algorithms and run in $O(m)$ time.

They both hash the incoming elements to $[n]$.

- SIMPLE-COUNT stores the minimum of the hashed values.
- SIMPLE-COUNT uses $O(\log n)$ bits of space.
- It gives us $\Pr(d \leq \hat{d}/3) \leq 1/3$ and $\Pr(d > 3\hat{d}) \leq 1/3$.

Summary

We saw two streaming algorithms for estimating the number of distinct items in a stream.

They are both one-pass algorithms and run in $O(m)$ time.

They both hash the incoming elements to $[n]$.

- Simple-count stores the minimum of the hashed values.
- Simple-count uses $O(\log n)$ bits of space.
- It gives us $\Pr(d \leq \hat{d}/3) \leq 1/3$ and $\Pr(d > 3\hat{d}) \leq 1/3$.
- Tidemark stores the maximum number of trailing zeros in the hashed values.

Summary

We saw two streaming algorithms for estimating the number of distinct items in a stream.

They are both one-pass algorithms and run in $O(m)$ time.

They both hash the incoming elements to $[n]$.

- SIMPLE-COUNT stores the minimum of the hashed values.
- SIMPLE-COUNT uses $O(\log n)$ bits of space.
- It gives us $\Pr(d \leq \hat{d}/3) \leq 1/3$ and $\Pr(d > 3\hat{d}) \leq 1/3$.
- TIDEMARK stores the maximum number of trailing zeros in the hashed values.
- TIDEMARK uses $O(\log \log n)$ bits of space.

Summary

We saw two streaming algorithms for estimating the number of distinct items in a stream.

They are both one-pass algorithms and run in $O(m)$ time.

They both hash the incoming elements to $[n]$.

- SIMPLE-COUNT stores the minimum of the hashed values.
- SIMPLE-COUNT uses $O(\log n)$ bits of space.
- It gives us $\Pr(d \leq \hat{d}/3) \leq 1/3$ and $\Pr(d > 3\hat{d}) \leq 1/3$.
- TIDEMARK stores the maximum number of trailing zeros in the hashed values.
- TIDEMARK uses $O(\log \log n)$ bits of space.
- It gives us $\Pr(\hat{d} \leq d/3) \leq \frac{\sqrt{2}}{3}$ and $\Pr(\hat{d} \geq 3d) \leq \frac{\sqrt{2}}{3}$

Summary

We saw two streaming algorithms for estimating the number of distinct items in a stream.

They are both one-pass algorithms and run in $O(m)$ time.

They both hash the incoming elements to $[n]$.

- SIMPLE-COUNT stores the minimum of the hashed values.
- SIMPLE-COUNT uses $O(\log n)$ bits of space.
- It gives us $\Pr(d \leq \hat{d}/3) \leq 1/3$ and $\Pr(d > 3\hat{d}) \leq 1/3$.
- TIDEMARK stores the maximum number of trailing zeros in the hashed values.
- TIDEMARK uses $O(\log \log n)$ bits of space.
- It gives us $\Pr(\hat{d} \leq d/3) \leq \frac{\sqrt{2}}{3}$ and $\Pr(\hat{d} \geq 3d) \leq \frac{\sqrt{2}}{3}$
- By performing k parallel iterations the probability of being outside the range $[d/3, 3d]$ can be made exponentially small.