# Topics in TCS

## Sparse recovery

Raphaël Clifford

# Sparse recovery

We will be working in the turnstile streaming model for this lecture.

# Sparse recovery

We will be working in the turnstile streaming model for this lecture.

We have seen how to estimate the frequency of tokens but not how to recover them from a sketch.

# Sparse recovery

We will be working in the turnstile streaming model for this lecture.

We have seen how to estimate the frequency of tokens but not how to recover them from a sketch.

If the number of tokens with non-zero frequency is at most $s$ then we say the stream is $s$-sparse. If the number of non-zero frequency tokens is exactly $s$ we say the stream is *strictly* $s$-sparse.

# Sparse recovery

We will be working in the turnstile streaming model for this lecture.

We have seen how to estimate the frequency of tokens but not how to recover them from a sketch.

If the number of tokens with non-zero frequency is at most $s$ then we say the stream is $s$-sparse. If the number of non-zero frequency tokens is exactly $s$ we say the stream is *strictly* $s$-sparse.

Our goal will be an $s$-sparse recovery algorithm which uses space not much larger than $s$.

# Sparse recovery

We will be working in the turnstile streaming model for this lecture.

We have seen how to estimate the frequency of tokens but not how to recover them from a sketch.

If the number of tokens with non-zero frequency is at most $s$ then we say the stream is $s$-sparse. If the number of non-zero frequency tokens is exactly $s$ we say the stream is *strictly* $s$-sparse.

Our goal will be an $s$-sparse recovery algorithm which uses space not much larger than $s$.

Imagine that at some point in a stream, there is exactly one token with non-zero frequency. How can we recover it?

# 1-sparse recovery - simple case

- For 1-sparse recovery, the idea is to maintain

$$\ell = \sum_{j=1}^{n} f_j \quad \text{and} \quad z = \sum_{j=1}^{n} j f_j$$

# 1-sparse recovery - simple case

- For 1-sparse recovery, the idea is to maintain

$$\ell = \sum_{j=1}^{n} f_j \quad \text{and} \quad z = \sum_{j=1}^{n} j f_j$$

- $\ell$ is the sum of the frequencies of tokens in the stream and $z$ is the weighted sum of those frequencies.

# 1-sparse recovery - simple case

- For 1-sparse recovery, the idea is to maintain

$$\ell = \sum_{j=1}^{n} f_j \quad \text{and} \quad z = \sum_{j=1}^{n} j f_j$$

- $\ell$ is the sum of the frequencies of tokens in the stream and $z$ is the weighted sum of those frequencies.
- Both $\ell$ and $z$ can be maintained in constant time per arriving token.

# 1-sparse recovery - simple case

- For 1-sparse recovery, the idea is to maintain

$$\ell = \sum_{j=1}^{n} f_j \quad \text{and} \quad z = \sum_{j=1}^{n} j f_j$$

- $\ell$ is the sum of the frequencies of tokens in the stream and $z$ is the weighted sum of those frequencies.
- Both $\ell$ and $z$ can be maintained in constant time per arriving token.
- If we know the stream is 1-sparse then the following simple deterministic algorithm works.

# 1-sparse recovery - simple case

- For 1-sparse recovery, the idea is to maintain

$$\ell = \sum_{j=1}^{n} f_j \quad \text{and} \quad z = \sum_{j=1}^{n} j f_j$$

- $\ell$ is the sum of the frequencies of tokens in the stream and $z$ is the weighted sum of those frequencies.
- Both $\ell$ and $z$ can be maintained in constant time per arriving token.
- If we know the stream is 1-sparse then the following simple deterministic algorithm works.
  1. If $\ell = 0$ there is nothing to return.

# 1-sparse recovery - simple case

- For 1-sparse recovery, the idea is to maintain

$$\ell = \sum_{j=1}^{n} f_j \quad \text{and} \quad z = \sum_{j=1}^{n} j f_j$$

- $\ell$ is the sum of the frequencies of tokens in the stream and $z$ is the weighted sum of those frequencies.
- Both $\ell$ and $z$ can be maintained in constant time per arriving token.
- If we know the stream is 1-sparse then the following simple deterministic algorithm works.
    1. If $\ell = 0$ there is nothing to return.
    2. Otherwise, $z/\ell$ is the identity of the token with non-zero frequency.

# 1-sparse recovery - simple case

- For 1-sparse recovery, the idea is to maintain

$$\ell = \sum_{j=1}^{n} f_j \quad \text{and} \quad z = \sum_{j=1}^{n} j f_j$$

- $\ell$ is the sum of the frequencies of tokens in the stream and $z$ is the weighted sum of those frequencies.
- Both $\ell$ and $z$ can be maintained in constant time per arriving token.
- If we know the stream is 1-sparse then the following simple deterministic algorithm works.
  1. If $\ell = 0$ there is nothing to return.
  2. Otherwise, $z/\ell$ is the identity of the token with non-zero frequency.
  3. We know the frequency of token $z/\ell$ is $\ell$.

# 1-sparse recovery - simple case

- For 1-sparse recovery, the idea is to maintain

$$\ell = \sum_{j=1}^{n} f_j \quad \text{and} \quad z = \sum_{j=1}^{n} j f_j$$

- $\ell$ is the sum of the frequencies of tokens in the stream and $z$ is the weighted sum of those frequencies.
- Both $\ell$ and $z$ can be maintained in constant time per arriving token.
- If we know the stream is 1-sparse then the following simple deterministic algorithm works.
  1. If $\ell = 0$ there is nothing to return.
  2. Otherwise, $z/\ell$ is the identity of the token with non-zero frequency.
  3. We know the frequency of token $z/\ell$ is $\ell$.
- If we don't know if the stream is 1-sparse it is much harder and we will have to use randomisation.

# 1-sparse recovery preparation

- We will need a prime field $\mathbb{F}_q$ which we can think of as being the integers modulo a prime $q$.

# 1-sparse recovery preparation

- We will need a prime field $\mathbb{F}_q$ which we can think of as being the integers modulo a prime $q$.
- For example consider $\mathbb{F}_3$. In this field $2 + 3 = 2, 2 \cdot 2 = 1, 3 \cdot 3 = 0$.

# 1-sparse recovery preparation

- We will need a prime field $\mathbb{F}_q$ which we can think of as being the integers modulo a prime $q$.
- For example consider $\mathbb{F}_3$. In this field $2 + 3 = 2, 2 \cdot 2 = 1, 3 \cdot 3 = 0$.
- We choose a prime $q$ such that $n^3 < q \leq 2n^3$. This is always possible by the Bertrand–Chebyshev theorem.

# 1-sparse recovery preparation

- We will need a prime field $\mathbb{F}_q$ which we can think of as being the integers modulo a prime $q$.
- For example consider $\mathbb{F}_3$. In this field $2 + 3 = 2, 2 \cdot 2 = 1, 3 \cdot 3 = 0$.
- We choose a prime $q$ such that $n^3 < q \leq 2n^3$. This is always possible by the Bertrand–Chebyshev theorem.
- We will use a *fingerprint*. This is in our case a randomised map from a vector to a much smaller *sketch*.

# 1-sparse recovery preparation

- We will need a prime field $\mathbb{F}_q$ which we can think of as being the integers modulo a prime $q$.
- For example consider $\mathbb{F}_3$. In this field $2 + 3 = 2, 2 \cdot 2 = 1, 3 \cdot 3 = 0$.
- We choose a prime $q$ such that $n^3 < q \leq 2n^3$. This is always possible by the Bertrand–Chebyshev theorem.
- We will use a *fingerprint*. This is in our case a randomised map from a vector to a much smaller *sketch*.
- Sketches should have the property that if two vectors are distinct then with high probability the sketches are distinct too. If two vectors are equal then our sketches will always be equal.

# 1-sparse recovery preparation

- We will need a prime field $\mathbb{F}_q$ which we can think of as being the integers modulo a prime $q$.
- For example consider $\mathbb{F}_3$. In this field $2+3=2, 2\cdot 2=1, 3\cdot 3=0$.
- We choose a prime $q$ such that $n^3 < q \leq 2n^3$. This is always possible by the Bertrand–Chebyshev theorem.
- We will use a *fingerprint*. This is in our case a randomised map from a vector to a much smaller *sketch*.
- Sketches should have the property that if two vectors are distinct then with high probability the sketches are distinct too. If two vectors are equal then our sketches will always be equal.
- We will use a polynomial sketch.

# The polynomial fingerprint

- We will compute $\ell = \sum_{j=1}^{n} f_j$ and $z = \sum_{j=1}^{n} j f_j$ as before.

# The polynomial fingerprint

- We will compute $\ell = \sum_{j=1}^{n} f_j$ and $z = \sum_{j=1}^{n} j f_j$ as before.

- We also compute $p = \sum_{j=1}^{n} c_j r^j$ for a random $r \in \mathbb{F}_q$. The value $c_j$ is the possibly negative count for token $j$.

# The polynomial fingerprint

- We will compute $\ell = \sum_{j=1}^{n} f_j$ and $z = \sum_{j=1}^{n} j f_j$ as before.

- We also compute $p = \sum_{j=1}^{n} c_j r^j$ for a random $r \in \mathbb{F}_q$. The value $c_j$ is the possibly negative count for token $j$.

- If there is exactly one non-zero frequency then $z/\ell$ is the identity of the token as before and $p = \ell r^{z/\ell}$ as all the other coefficients will be zero.

# The polynomial fingerprint

- We will compute $\ell = \sum_{j=1}^{n} f_j$ and $z = \sum_{j=1}^{n} jf_j$ as before.
- We also compute $p = \sum_{j=1}^{n} c_j r^j$ for a random $r \in \mathbb{F}_q$. The value $c_j$ is the possibly negative count for token $j$.
- If there is exactly one non-zero frequency then $z/\ell$ is the identity of the token as before and $p = \ell r^{z/\ell}$ as all the other coefficients will be zero.
- Otherwise, we show that $p \neq \ell r^{z/\ell}$ with high probability.

# 1-sparse recovery algorithm

Define $\boldsymbol{e}_i$ to be an all zero vector except for a 1 at index $i$.

```
initialise (ℓ, z, p) = (0, 0, 0)
choose r to be a uniform random element of 𝔽_q

1-SPARSE(j, c)                                  # token, count
set ℓ = ℓ + c
set z = z + cj
set p = p + cr^j                                # fingerprint

OUTPUT
if ℓ = z = p = 0 return 0
     return f = 0
     else if z/ℓ ∉ [n]
          return ‖f‖_0 > 1
     else if p ≠ ℓr^(z/ℓ)
          return ‖f‖_0 > 1
     else
          return f = ℓe_(z/ℓ)
```

# 1-sparse recovery - example

1-SPARSE$(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.

# 1-sparse recovery - example

1-SPARSE$(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle (2,3), (1,-2), (2,-2), (1,2) \rangle$.

# 1-sparse recovery - example

$1\text{-SPARSE}(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle(2,3),(1,-2),(2,-2),(1,2)\rangle$.
- Choose a random $r \in \{0,\ldots,10\}$. Say $r = 5$.

# 1-sparse recovery - example

1-SPARSE$(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle (2, 3), (1, -2), (2, -2), (1, 2) \rangle$.
- Choose a random $r \in \{0, \ldots, 10\}$. Say $r = 5$.

- $\ell$ is updated to $3, 1, -1, 1$ in turn.

# 1-sparse recovery - example

$1\text{-}\textsc{sparse}(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle(2,3),(1,-2),(2,-2),(1,2)\rangle$.
- Choose a random $r \in \{0,\ldots,10\}$. Say $r = 5$.

- $\ell$ is updated to $3, 1, -1, 1$ in turn.
- $z$ is updated to $6, 4, 0, 2$ in turn.

# 1-sparse recovery - example

$1\text{-SPARSE}(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle(2,3),(1,-2),(2,-2),(1,2)\rangle$.
- Choose a random $r \in \{0,\ldots,10\}$. Say $r = 5$.
- $\ell$ is updated to $3, 1, -1, 1$ in turn.
- $z$ is updated to $6, 4, 0, 2$ in turn.
- $p$ is updated to $3 \cdot 5^2 = 75$, $75 + (-2)5^1 = 65$, $65 + (-2)5^2 = 15, 15 + 2 \cdot 5^1 = 25$.

# 1-sparse recovery - example

1-SPARSE$(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle (2,3), (1,-2), (2,-2), (1,2) \rangle$.
- Choose a random $r \in \{0, \dots, 10\}$. Say $r = 5$.
- $\ell$ is updated to $3, 1, -1, 1$ in turn.
- $z$ is updated to $6, 4, 0, 2$ in turn.
- $p$ is updated to $3 \cdot 5^2 = 75$, $75 + (-2)5^1 = 65$, $65 + (-2)5^2 = 15, 15 + 2 \cdot 5^1 = 25$.
- In $\mathbb{F}_{11}$ the values of $p$ are in turn: $9, 10, 4, 3$.

# 1-sparse recovery - example

$1\text{-SPARSE}(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle(2, 3), (1, -2), (2, -2), (1, 2)\rangle$.
- Choose a random $r \in \{0, \ldots, 10\}$. Say $r = 5$.

- $\ell$ is updated to $3, 1, -1, 1$ in turn.
- $z$ is updated to $6, 4, 0, 2$ in turn.
- $p$ is updated to $3 \cdot 5^2 = 75$, $75 + (-2)5^1 = 65$, $65 + (-2)5^2 = 15, 15 + 2 \cdot 5^1 = 25$.
- In $\mathbb{F}_{11}$ the values of $p$ are in turn: $9, 10, 4, 3$.
- Now we check $\ell \neq 0$ ✓

# 1-sparse recovery - example

1-$\textsc{sparse}(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle(2,3),(1,-2),(2,-2),(1,2)\rangle$.
- Choose a random $r \in \{0,\ldots,10\}$. Say $r = 5$.
- $\ell$ is updated to $3, 1, -1, 1$ in turn.
- $z$ is updated to $6, 4, 0, 2$ in turn.
- $p$ is updated to $3 \cdot 5^2 = 75$, $75 + (-2)5^1 = 65$, $65 + (-2)5^2 = 15, 15 + 2 \cdot 5^1 = 25$.
- In $\mathbb{F}_{11}$ the values of $p$ are in turn: $9, 10, 4, 3$.
- Now we check $\ell \neq 0$ ✓
- Check $z/\ell = 2/1 = 2 \in [2]$ ✓

# 1-sparse recovery - example

$1\text{-}\textsc{sparse}(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle(2,3),(1,-2),(2,-2),(1,2)\rangle$.
- Choose a random $r \in \{0,\ldots,10\}$. Say $r = 5$.
- $\ell$ is updated to $3, 1, -1, 1$ in turn.
- $z$ is updated to $6, 4, 0, 2$ in turn.
- $p$ is updated to $3 \cdot 5^2 = 75$, $75 + (-2)5^1 = 65$, $65 + (-2)5^2 = 15, 15 + 2 \cdot 5^1 = 25$.
- In $\mathbb{F}_{11}$ the values of $p$ are in turn: $9, 10, 4, 3$.
- Now we check $\ell \neq 0$ ✓
- Check $z/\ell = 2/1 = 2 \in [2]$ ✓
- Check $p = \ell r^{z/\ell} = 1 \cdot 5^2 = 3 \bmod 11$. ✓

# 1-sparse recovery - example

1-sparse$(j, c)$
set $\ell = \ell + c$
set $z = z + cj$
set $p = p + cr^j$

- Let $n = 2$ and $q = 11$.
- Consider stream $\langle(2, 3), (1, -2), (2, -2), (1, 2)\rangle$.
- Choose a random $r \in \{0, \ldots, 10\}$. Say $r = 5$.
- $\ell$ is updated to $3, 1, -1, 1$ in turn.
- $z$ is updated to $6, 4, 0, 2$ in turn.
- $p$ is updated to $3 \cdot 5^2 = 75$, $75 + (-2)5^1 = 65$, $65 + (-2)5^2 = 15, 15 + 2 \cdot 5^1 = 25$.
- In $\mathbb{F}_{11}$ the values of $p$ are in turn: $9, 10, 4, 3$.
- Now we check $\ell \neq 0$ ✓
- Check $z/\ell = 2/1 = 2 \in [2]$ ✓
- Check $p = \ell r^{z/\ell} = 1 \cdot 5^2 = 3 \bmod 11$. ✓
- Output: $\ell \boldsymbol{e}_{z/\ell} = (0, 1)$

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.
- In that case $\ell = f_i, z = i\ell$ and so $z/\ell = i \in [n]$.

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.
- In that case $\ell = f_i, z = i\ell$ and so $z/\ell = i \in [n]$.
- Therefore $p = f_i r^i = \ell r^{z/\ell}$ as required.

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.
- In that case $\ell = f_i, z = i\ell$ and so $z/\ell = i \in [n]$.
- Therefore $p = f_i r^i = \ell r^{z/\ell}$ as required.
- If $\boldsymbol{f} = \mathbf{0}$ then $\ell = z = p = 0$ and so the algorithm returns $\mathbf{0}$ correctly.

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.
- In that case $\ell = f_i, z = i\ell$ and so $z/\ell = i \in [n]$.
- Therefore $p = f_i r^i = \ell r^{z/\ell}$ as required.
- If $\boldsymbol{f} = \mathbf{0}$ then $\ell = z = p = 0$ and so the algorithm returns $\mathbf{0}$ correctly.
- If $\boldsymbol{f}$ has exactly one non-zero frequency token then the algorithm returns $\ell \boldsymbol{e}_{z/\ell}$ correctly.

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.
- In that case $\ell = f_i, z = i\ell$ and so $z/\ell = i \in [n]$.
- Therefore $p = f_i r^i = \ell r^{z/\ell}$ as required.
- If $\boldsymbol{f} = \mathbf{0}$ then $\ell = z = p = 0$ and so the algorithm returns $\mathbf{0}$ correctly.
- If $\boldsymbol{f}$ has exactly one non-zero frequency token then the algorithm returns $\ell \boldsymbol{e}_{z/\ell}$ correctly.
- No false negatives.

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.
- In that case $\ell = f_i, z = i\ell$ and so $z/\ell = i \in [n]$.
- Therefore $p = f_i r^i = \ell r^{z/\ell}$ as required.
- If $\boldsymbol{f} = \mathbf{0}$ then $\ell = z = p = 0$ and so the algorithm returns $\mathbf{0}$ correctly.
- If $\boldsymbol{f}$ has exactly one non-zero frequency token then the algorithm returns $\ell \boldsymbol{e}_{z/\ell}$ correctly.
- No false negatives.
- What about if $\boldsymbol{f}$ is not 1-sparse? Could we get false-positives?

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.
- In that case $\ell = f_i, z = i\ell$ and so $z/\ell = i \in [n]$.
- Therefore $p = f_i r^i = \ell r^{z/\ell}$ as required.
- If $\boldsymbol{f} = \mathbf{0}$ then $\ell = z = p = 0$ and so the algorithm returns $\mathbf{0}$ correctly.
- If $\boldsymbol{f}$ has exactly one non-zero frequency token then the algorithm returns $\ell \boldsymbol{e}_{z/\ell}$ correctly.
- No false negatives.
- What about if $\boldsymbol{f}$ is not 1-sparse? Could we get false-positives?
- Yes. It is possible that $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$ but the stream is not 1-sparse.

# 1-sparse recovery - analysis

- Consider the case where $\boldsymbol{f}$ is strictly 1-sparse where $i$ is the token with non-zero frequency.
- In that case $\ell = f_i, z = i\ell$ and so $z/\ell = i \in [n]$.
- Therefore $p = f_i r^i = \ell r^{z/\ell}$ as required.
- If $\boldsymbol{f} = \mathbf{0}$ then $\ell = z = p = 0$ and so the algorithm returns $\mathbf{0}$ correctly.
- If $\boldsymbol{f}$ has exactly one non-zero frequency token then the algorithm returns $\ell \boldsymbol{e}_{z/\ell}$ correctly.
- No false negatives.
- What about if $\boldsymbol{f}$ is not 1-sparse? Could we get false-positives?
- Yes. It is possible that $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$ but the stream is not 1-sparse.
- How likely are we to get false-positives?

# 1-sparse recovery - how likely are false positives?

**Number of roots of a polynomial**

Over any field, a nonzero polynomial of degree $d$ has at most $d$ roots

# 1-sparse recovery - how likely are false positives?

**Number of roots of a polynomial**

Over any field, a nonzero polynomial of degree $d$ has at most $d$ roots

- We have that $p = \sum_{j=1}^{n} f_j r^j = q(r)$ where $q(r)$ is a polynomial over the finite field $\mathbb{F}_q$.

# 1-sparse recovery - how likely are false positives?

**Number of roots of a polynomial**

Over any field, a nonzero polynomial of degree $d$ has at most $d$ roots

- We have that $p = \sum_{j=1}^{n} f_j r^j = q(r)$ where $q(r)$ is a polynomial over the finite field $\mathbb{F}_q$.
- If the stream is not 1-sparse then we know that a false positive can only occur when $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$.

# 1-sparse recovery - how likely are false positives?

**Number of roots of a polynomial**

Over any field, a nonzero polynomial of degree $d$ has at most $d$ roots

- We have that $p = \sum_{j=1}^{n} f_j r^j = q(r)$ where $q(r)$ is a polynomial over the finite field $\mathbb{F}_q$.
- If the stream is not 1-sparse then we know that a false positive can only occur when $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$.
- To handle both these cases at once, define

$$i = \begin{cases} 0, & \text{if } \ell = z = 0 \text{ or } z/\ell \notin [n] \\ z/\ell, & \text{otherwise} \end{cases}$$

# 1-sparse recovery - how likely are false positives?

**Number of roots of a polynomial**

Over any field, a nonzero polynomial of degree $d$ has at most $d$ roots

- We have that $p = \sum_{j=1}^{n} f_j r^j = q(r)$ where $q(r)$ is a polynomial over the finite field $\mathbb{F}_q$.
- If the stream is not 1-sparse then we know that a false positive can only occur when $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$.
- To handle both these cases at once, define

$$i = \begin{cases} 0, & \text{if } \ell = z = 0 \text{ or } z/\ell \notin [n] \\ z/\ell, & \text{otherwise} \end{cases}$$

- Now we have that a false positive occurs only when $r$ is a root of the polynomial $q(x) - \ell x^i$.

# 1-sparse recovery - how likely are false positives?

**Number of roots of a polynomial**

Over any field, a nonzero polynomial of degree $d$ has at most $d$ roots

- We have that $p = \sum_{j=1}^{n} f_j r^j = q(r)$ where $q(r)$ is a polynomial over the finite field $\mathbb{F}_q$.
- If the stream is not 1-sparse then we know that a false positive can only occur when $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$.
- To handle both these cases at once, define

$$i = \begin{cases} 0, & \text{if } \ell = z = 0 \text{ or } z/\ell \notin [n] \\ z/\ell, & \text{otherwise} \end{cases}$$

- Now we have that a false positive occurs only when $r$ is a root of the polynomial $q(x) - \ell x^i$.
- But $q(x) - \ell x^{z/\ell}$ has degree at most $n$ and hence at most $n$ roots and so

$$\Pr(r \text{ is a root of } q(x) - \ell x^{z/\ell}) \le \frac{n}{|\mathbb{F}_q|} \in O\left(\frac{1}{n^2}\right)$$

# 1-sparse recovery - summary and time/space

- The 1-sparse algorithm always gives the correct answer for a 1-sparse stream.

# 1-sparse recovery - summary and time/space

- The 1-sparse algorithm always gives the correct answer for a 1-sparse stream.

- We get a false positive if the stream is not 1-sparse and either $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$. This occurs with probability at most $O(1/n^2)$.

# 1-sparse recovery - summary and time/space

- The 1-sparse algorithm always gives the correct answer for a 1-sparse stream.

- We get a false positive if the stream is not 1-sparse and either $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$. This occurs with probability at most $O(1/n^2)$.

- Each $(j, c)$ pair is processed in constant time so the total running time is $O(m)$.

# 1-sparse recovery - summary and time/space

- The 1-sparse algorithm always gives the correct answer for a 1-sparse stream.

- We get a false positive if the stream is not 1-sparse and either $\ell = z = p = 0$ or $z/\ell \in [n]$ and $p = \ell r^{z/\ell}$. This occurs with probability at most $O(1/n^2)$.

- Each $(j, c)$ pair is processed in constant time so the total running time is $O(m)$.

- If $M$ is the largest frequency of any item, the total space is $O(\log n + \log M)$ bits. This is because $|\ell| \leq nM$ and $|z| \leq n^2 M$ and $p, r \in \mathbb{F}_q$ with $q \leq 2n^3$.

# $s$-sparse recovery

- Let $s \ll n$. We want to recover all the tokens with non-zero frequency provided there at most $s$ of them and or report that there are too many otherwise.

# $s$-sparse recovery

- Let $s \ll n$. We want to recover all the tokens with non-zero frequency provided there at most $s$ of them and or report that there are too many otherwise.

- The overall idea is to randomly map the tokens into $2s$ streams.

# $s$-sparse recovery

- Let $s \ll n$. We want to recover all the tokens with non-zero frequency provided there at most $s$ of them and or report that there are too many otherwise.

- The overall idea is to randomly map the tokens into $2s$ streams.

- If the stream is $s$-sparse there will be a good chance that individual streams will be 1-sparse.

# $s$-sparse recovery

- Let $s \ll n$. We want to recover all the tokens with non-zero frequency provided there at most $s$ of them and or report that there are too many otherwise.

- The overall idea is to randomly map the tokens into $2s$ streams.

- If the stream is $s$-sparse there will be a good chance that individual streams will be 1-sparse.

- We run our 1-sparse detection and recovery algorithm on each stream.

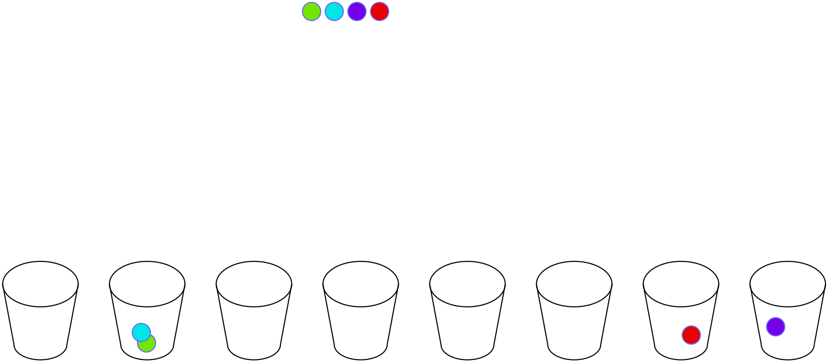
## $s$-sparse recovery

- Let $s \ll n$. We want to recover all the tokens with non-zero frequency provided there at most $s$ of them and or report that there are too many otherwise.
- The overall idea is to randomly map the tokens into $2s$ streams.
- If the stream is $s$-sparse there will be a good chance that individual streams will be 1-sparse.
- We run our 1-sparse detection and recovery algorithm on each stream.
- We repeat the whole process to decrease the error probability.

# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

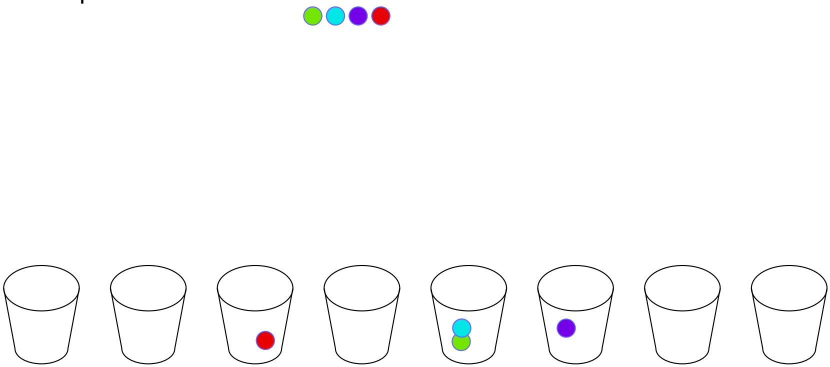
Attempt 1:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

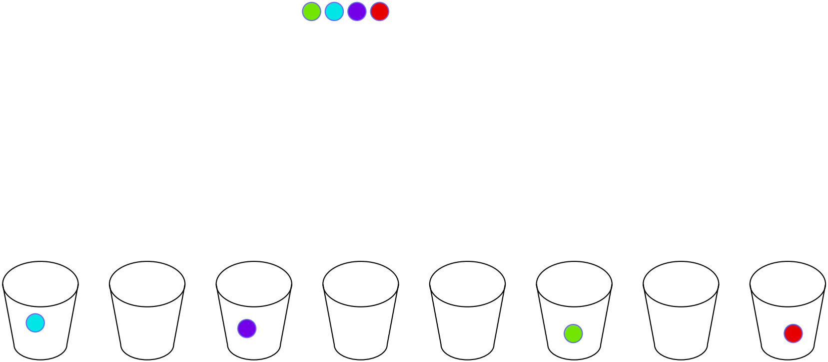
Attempt 2:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

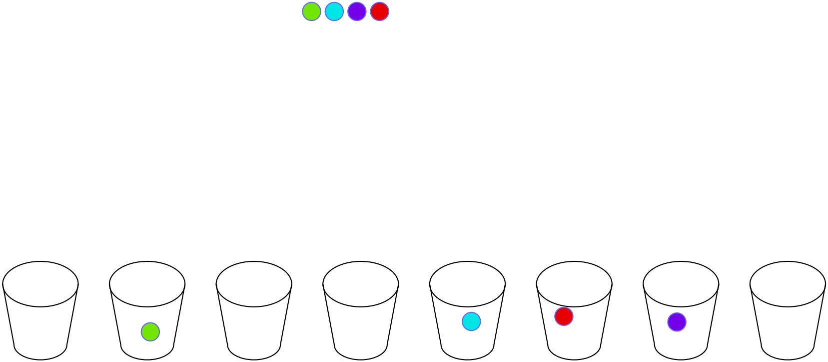
Attempt 3:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

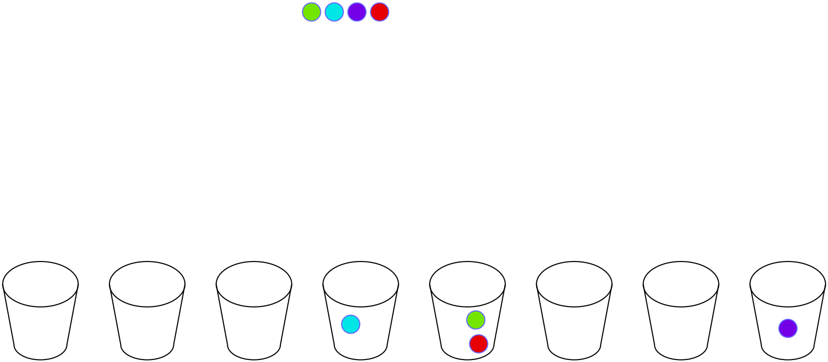
Attempt 4:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

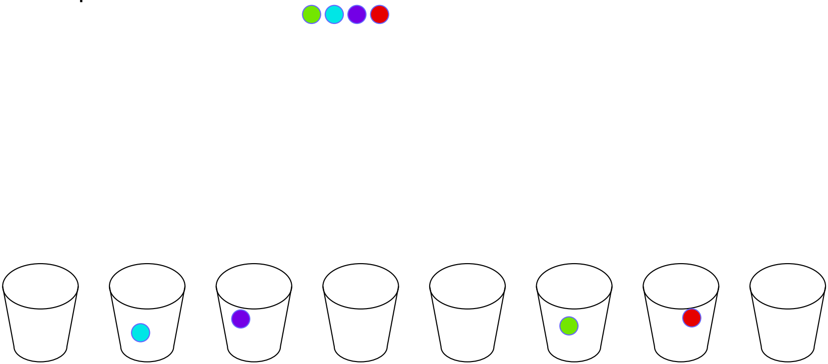
Attempt 5:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

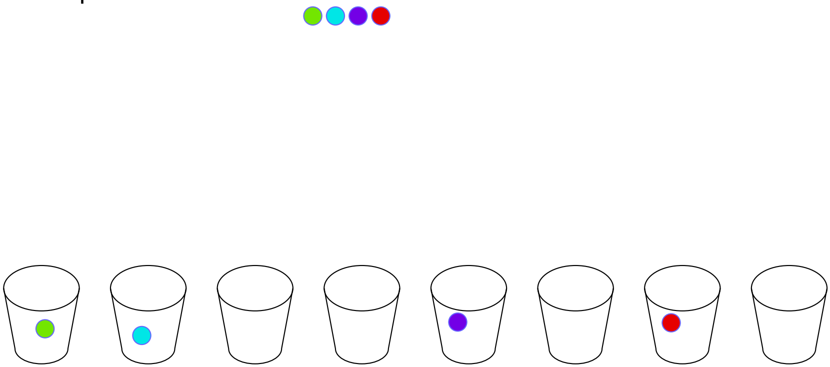
Attempt 6:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

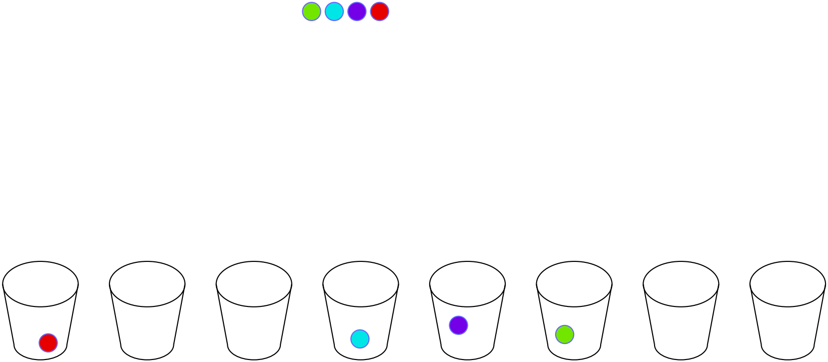
Attempt 7:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

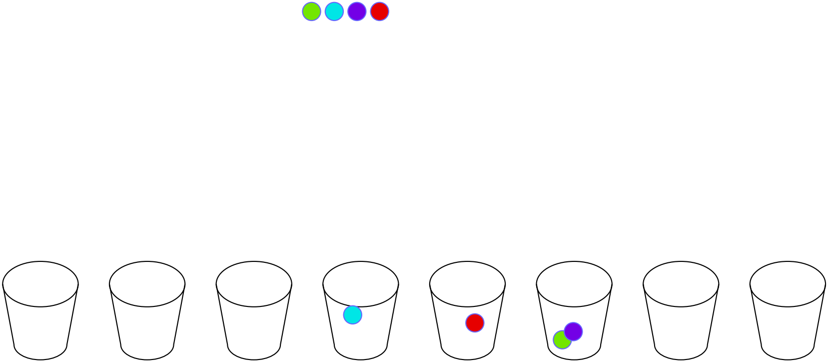
Attempt 8:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

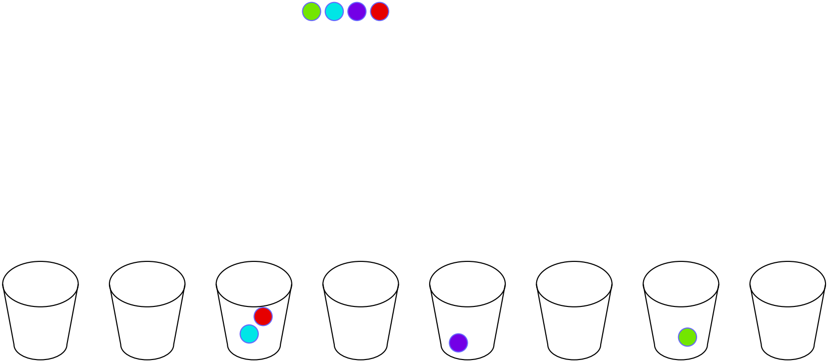
Attempt 9:



# Throwing $s$ balls in $2s$ bins

What happens when you throw $s$ balls into $2s$ bins?

Attempt 10:



# $s$-sparse recovery

```
initialise t = ⌈log(s/δ)⌉
initialise D[1...t][1...2s] = 0
choose t independent hash functions h_1,...,h_t : [n] → [2s]

s-SPARSE(j, c)                              # c can be negative
for each i ∈ [t]
     update D[i, h_i(j)] with token, count pair (j, c)
```

- `update` runs one step of the 1-sparse recovery and detection algorithm with the new incoming token pair.

## $s$-sparse recovery

```
initialise t = ⌈log(s/δ)⌉
initialise D[1...t][1...2s] = 0
choose t independent hash functions h_1,...,h_t : [n] → [2s]

s-SPARSE(j, c)                          # c can be negative
for each i ∈ [t]
     update D[i, h_i(j)] with token, count pair (j, c)
```

- update runs one step of the 1-sparse recovery and detection algorithm with the new incoming token pair.
- The token $j$ can be mapped to any of the $2s$ columns of $D$. Each column represents a different substream.

# $s$-sparse recovery

```
initialise t = ⌈log(s/δ)⌉
initialise D[1...t][1...2s] = 0
choose t independent hash functions h_1, ..., h_t : [n] → [2s]

s-SPARSE(j, c)                         # c can be negative
for each i ∈ [t]
     update D[i, h_i(j)] with token, count pair (j, c)
```

- `update` runs one step of the 1-sparse recovery and detection algorithm with the new incoming token pair.
- The token $j$ can be mapped to any of the $2s$ columns of $D$. Each column represents a different substream.
- $D[i, j]$ stores the variables and result for an instance of the 1-sparse recovery and detection algorithm.

# $s$-sparse recovery

```
initialise t = ⌈log(s/δ)⌉
initialise D[1...t][1...2s] = 0
choose t independent hash functions h_1, ..., h_t : [n] → [2s]

s-SPARSE(j, c)                          # c can be negative
for each i ∈ [t]
     update D[i, h_i(j)] with token, count pair (j, c)
```

- `update` runs one step of the 1-sparse recovery and detection algorithm with the new incoming token pair.
- The token $j$ can be mapped to any of the $2s$ columns of $D$. Each column represents a different substream.
- $D[i, j]$ stores the variables and result for an instance of the 1-sparse recovery and detection algorithm.
- The $t$ rows of $D$ are for the $t$ independent hash functions.

# $s$-sparse recovery - output

```
initialise t = ⌈log(s/δ)⌉
initialise D[1...t][1...2s] = 0
choose t independent hash functions h_1,...,h_t : [n] → [2s]

s-SPARSE(j, c)                                # c can be negative
for each i ∈ [t]
     update D[i, h_i(j)] with pair (j, c)
```

For the final output we take the following steps:

- For each $i \in [t], k \in [2s]$

# $s$-sparse recovery - output

```
initialise t = ⌈log(s/δ)⌉
initialise D[1...t][1...2s] = 0
choose t independent hash functions h_1,...,h_t : [n] → [2s]

s-SPARSE(j, c)                           # c can be negative
for each i ∈ [t]
     update D[i, h_i(j)] with pair (j, c)
```

For the final output we take the following steps:

- For each $i \in [t], k \in [2s]$
  - if $D[i, k]$ reports success then store the imputed token/index if it isn't contradicted by a previously stored index for that stream.

# $s$-sparse recovery - output

```
initialise t = ⌈log(s/δ)⌉
initialise D[1...t][1...2s] = 0
choose t independent hash functions h_1, ..., h_t : [n] → [2s]

s-SPARSE(j, c)                              # c can be negative
for each i ∈ [t]
     update D[i, h_i(j)] with pair (j, c)
```

For the final output we take the following steps:

- For each $i \in [t], k \in [2s]$
  - if $D[i, k]$ reports success then store the imputed token/index if it isn't contradicted by a previously stored index for that stream.
  - if the total number of tokens/indices stored is greater than $s$, then abort.

# $s$-sparse recovery - output

```
initialise t = ⌈log(s/δ)⌉
initialise D[1...t][1...2s] = 0
choose t independent hash functions h_1,...,h_t : [n] → [2s]

s-SPARSE(j, c)                           # c can be negative
for each i ∈ [t]
     update D[i, h_i(j)] with pair (j, c)
```

For the final output we take the following steps:

- For each $i \in [t], k \in [2s]$
  - if $D[i, k]$ reports success then store the imputed token/index if it isn't contradicted by a previously stored index for that stream.
  - if the total number of tokens/indices stored is greater than $s$, then abort.

- Output all frequency/token pairs inferred from the stored information.

## $s$-sparse recovery - analysis

- The $s$-sparse algorithm correctly outputs $\boldsymbol{f}$ if both of the following happens:

# $s$-sparse recovery - analysis

- The $s$-sparse algorithm correctly outputs $\boldsymbol{f}$ if both of the following happens:

[$\mathrm{SR}_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.

# $s$-sparse recovery - analysis

- The $s$-sparse algorithm correctly outputs $\boldsymbol{f}$ if both of the following happens:

[SR$_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.
[SR$_2$] None of the $2st$ entries in $D$ give a false positive.

## $s$-sparse recovery - analysis

- The $s$-sparse algorithm correctly outputs $\boldsymbol{f}$ if both of the following happens:

[$\mathrm{SR}_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.
[$\mathrm{SR}_2$] None of the $2st$ entries in $D$ give a false positive.

# $s$-sparse recovery - analysis

- The $s$-sparse algorithm correctly outputs $\boldsymbol{f}$ if both of the following happens:

[SR$_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.
[SR$_2$] None of the $2st$ entries in $D$ give a false positive.

Proof for SR$_1$. Consider a particular item $j \in \operatorname{supp} \boldsymbol{f}$. For each $i \in [t]$, let $\boldsymbol{\sigma}_i(j)$ be the substream generated by $h_i$ containing elements of the form $(j, c)$. Note that $f_j \neq 0$ necessarily for this stream.

# $s$-sparse recovery - analysis

- The $s$-sparse algorithm correctly outputs $\boldsymbol{f}$ if both of the following happens:

[SR$_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.
[SR$_2$] None of the $2st$ entries in $D$ give a false positive.

Proof for SR$_1$. Consider a particular item $j \in \operatorname{supp} \boldsymbol{f}$. For each $i \in [t]$, let $\boldsymbol{\sigma}_i(j)$ be the substream generated by $h_i$ containing elements of the form $(j, c)$. Note that $f_j \neq 0$ necessarily for this stream.

$$
\begin{aligned}
\Pr(\boldsymbol{\sigma}_i(j) \text{ is not 1-sparse}) &= \Pr(\exists j' \in \operatorname{supp} \boldsymbol{f} : j' \neq j \wedge h_i(j') = h_i(j)) \\
&\leq \sum_{\substack{j' \in \operatorname{supp} \boldsymbol{f} \\ j \neq j'}} \Pr(h_i(j') = h_i(j)) \\
&\leq \sum_{\substack{j' \in \operatorname{supp} \boldsymbol{f} \\ j \neq j'}} \frac{1}{2s} \leq \frac{1}{2}
\end{aligned}
$$

# $s$-sparse recovery - analysis

- The $s$-sparse algorithm correctly outputs $\boldsymbol{f}$ if both of the following happens:

[SR$_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.
[SR$_2$] None of the $2st$ entries in $D$ give a false positive.

Proof for SR$_1$. Consider a particular item $j \in \operatorname{supp} \boldsymbol{f}$. For each $i \in [t]$, let $\boldsymbol{\sigma}_i(j)$ be the substream generated by $h_i$ containing elements of the form $(j, c)$. Note that $f_j \neq 0$ necessarily for this stream.

$$\begin{aligned}
\Pr(\boldsymbol{\sigma}_i(j) \text{ is not 1-sparse}) &= \Pr(\exists j' \in \operatorname{supp} \boldsymbol{f} : j' \neq j \wedge h_i(j') = h_i(j)) \\
&\leq \sum_{\substack{j' \in \operatorname{supp} \boldsymbol{f} \\ j \neq j'}} \Pr(h_i(j') = h_i(j)) \\
&\leq \sum_{\substack{j' \in \operatorname{supp} \boldsymbol{f} \\ j \neq j'}} \frac{1}{2s} \leq \frac{1}{2}
\end{aligned}$$

$\Pr(\text{SR}_1 \text{ fails for item } j) = \prod_{i=1}^{t} \Pr(\boldsymbol{\sigma}_i(j) \text{ is not 1-sparse}) \leq \left(\frac{1}{2}\right)^t \leq \frac{\delta}{s}$

# End of $s$-sparse proof

[$\mathrm{SR}_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.

[$\mathrm{SR}_2$] None of the $2st$ entries in $D$ give a false positive.

- We have that $\Pr(\mathrm{SR}_1 \text{ fails for item } j) \leq \frac{\delta}{s}$.

# End of $s$-sparse proof

[$\mathrm{SR}_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.

[$\mathrm{SR}_2$] None of the $2st$ entries in $D$ give a false positive.

- We have that $\Pr(\mathrm{SR}_1 \text{ fails for item } j) \leq \frac{\delta}{s}$.
- By a union bound over tokens in $\operatorname{supp} \boldsymbol{f}$,

$$\Pr(\mathrm{SR}_1 \text{ fails for any item}) \leq |\operatorname{supp} \boldsymbol{f}| \cdot \frac{\delta}{s} \leq \delta.$$

# End of $s$-sparse proof

[$\mathrm{SR}_1$] Every $j \in \operatorname{supp} \boldsymbol{f}$ is in at least one strictly 1-sparse substream.

[$\mathrm{SR}_2$] None of the $2st$ entries in $D$ give a false positive.

- We have that $\Pr(\mathrm{SR}_1 \text{ fails for item } j) \leq \frac{\delta}{s}$.
- By a union bound over tokens in $\operatorname{supp} \boldsymbol{f}$,

$$\Pr(\mathrm{SR}_1 \text{ fails for any item}) \leq |\operatorname{supp} \boldsymbol{f}| \cdot \frac{\delta}{s} \leq \delta.$$

- By another union bound over the entries of $D$,

$$\Pr(\mathrm{SR}_2 \text{ fails}) \leq 2st \cdot O\left(\frac{1}{n^2}\right) \in o(1),$$

because $st \leq n$. By yet another union bound the probability that the recovery fails is at most $\delta + o(1)$.

# $s$-sparse recovery - Summary

- The $s$-sparse recovery algorithm errs with probability of error at most $\delta + o(1)$.

# $s$-sparse recovery - Summary

- The $s$-sparse recovery algorithm errs with probability of error at most $\delta + o(1)$.

- The overall space used is $O(st(\log n + \log M)$ bits. This equals $O(s(\log s + \log \delta^{-1})(\log n + \log M))$ bits.

# $s$-sparse recovery - Summary

- The $s$-sparse recovery algorithm errs with probability of error at most $\delta + o(1)$.

- The overall space used is $O(st(\log n + \log M)$ bits. This equals $O(s(\log s + \log \delta^{-1})(\log n + \log M))$ bits.

- The running time is $O(t) = O(\log s + \log \delta^{-1})$ per arriving token and $O(tk) = O(s(\log s + \log \delta^{-1}))$ time to report the results.

# $s$-sparse recovery - Summary

- The $s$-sparse recovery algorithm errs with probability of error at most $\delta + o(1)$.

- The overall space used is $O(st(\log n + \log M)$ bits. This equals $O(s(\log s + \log \delta^{-1})(\log n + \log M))$ bits.

- The running time is $O(t) = O(\log s + \log \delta^{-1})$ per arriving token and $O(tk) = O(s(\log s + \log \delta^{-1}))$ time to report the results.

- How can we check if the stream really was $s$-sparse?