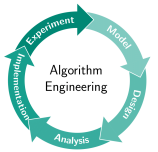
Sketches of Sets, Vectors and Graphs

Stefan Walzer, Ragnar Groot Koerkamp, Stefan Hermann | compiled on 22. Juni 2026



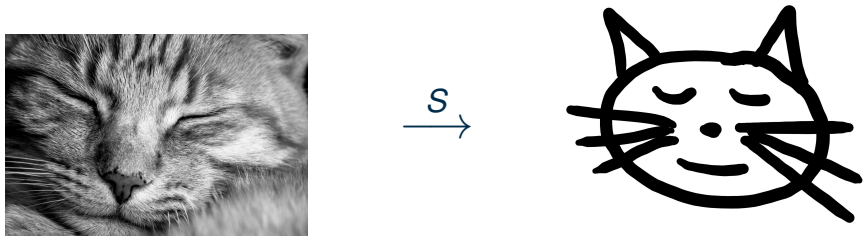
What is Sketching?

Definition

A sketch $S(X)$ of some data X arises from a (randomised) sketching function S.

Design Principles

- **small.** $S(X)$ requires *little memory*.
- **fast.** $S(X)$ can be updated quickly.
- **informative.** Certain queries regarding X can be answered using $S(X)$.
- **quantifiably lossy.** Queries on sketches may fail (with quantifiable error probability) or return imprecise answers (with quantifiable error bound).



Intuition: lossy representation that preserves relevant information

Sketches of Sets, Vectors and Graphs

1. Set Sketches...

... for Cardinality Estimation
... for Membership Estimation

2. Vector Sketches and the Turnstile Model

... for $\|x\|_2$-Norm Estimation
... for Heavy Hitter Detection

3. Set Sketches for Sparse Recovery

Warm-Up: 1-Sparse Recovery
n-Sparse Recovery
Getting Any Element
Application: Graph Sketch for Spanning Tree Computation

Set Sketching

Setting

U is a universe and $S : 2^U \to R$ a sketching function for some range R.

Constructing and Updating the Sketch

- **construct**: Given $X \subseteq U$, compute $S(X)$
- **insert**: Given $S(X)$ and $y \in U$, compute $S(X \cup \{y\})$
- **delete**: Given $S(X)$ and $y \in X$, compute $S(X \setminus \{y\})$
- **merge**: Given $S(X)$ and $S(Y)$, compute $S(X \cup Y)$

(only some of these may be supported)

Queries we will consider:

- Cardinality: Estimate $|X|$ from $S(X)$.
- Membership: Estimate $[x \in X]$ from $S(X)$ and x.

Set Sketch for Cardinality Estimation

Sketch Definition and Purpose

Let $h : U \to \{0,1\}^w$ be a random hash function and

$$S : \begin{cases} 2^U \setminus \{\emptyset\} & \to \mathbb{N}_0 \\ X & \mapsto \max_{x \in X} \text{countLeadingZeroes}(h(x)) \end{cases}$$

Goal: Estimate $|X|$ as $2^{S(X)}$.
Space: $\mathcal{O}(\log \log |X|)$ bits in expectation.
Updates: insert ✓, merge ✓

Example

$x \in X$	$h(x)$ in binary	leading zeroes
E	**00**101...	2
X	10111...	0
A	**000**10...	**3**
M	10110...	0
P	10110...	0
L	**0**1011...	1
E	**00**101...	2

largest number of leading zeroes is $S(X) = 3$.
$\Rightarrow$ estimate $|X| \approx 2^3 = 8$.

Set Sketch for Cardinality Estimation

Sketch Definition and Purpose

Let $h : U \to \{0,1\}^w$ be a random hash function and

$$S : \begin{cases} 2^U \setminus \{\emptyset\} & \to \mathbb{N}_0 \\ X & \mapsto \max_{x \in X} \text{countLeadingZeroes}(h(x)) \end{cases}$$

Goal: Estimate $|X|$ as $2^{S(X)}$.
Space: $\mathcal{O}(\log \log |X|)$ bits in expectation.
Updates: insert ✓, merge ✓

Intuition

$|X|$ elements have ≥ 0 leading zeroes
$\approx |X|/2$ elements have ≥ 1 leading zero
$\approx |X|/4$ elements have ≥ 2 leading zeroes
...
$\approx |X|/2^k$ elements have $\geq k$ leading zeroes
$\hookrightarrow$ highest number of leading zeroes is $\approx \log |X|$.

Example

$x \in X$	$h(x)$ in binary	leading zeroes
E	**00**101...	2
X	10111...	0
A	**000**10...	**3**
M	10110...	0
P	10110...	0
L	**0**1011...	1
E	**00**101...	2

largest number of leading zeroes is $S(X) = 3$.
$\Rightarrow$ estimate $|X| \approx 2^3 = 8$.

Set Sketch for Cardinality Estimation

Sketch Definition and Purpose

Let $h : U \to \{0,1\}^w$ be a random hash function and

$$S : \begin{cases} 2^U \setminus \{\emptyset\} & \to \mathbb{N}_0 \\ X & \mapsto \max_{x \in X} \text{countLeadingZeroes}(h(x)) \end{cases}$$

Goal: Estimate $|X|$ as $2^{S(X)}$.
Space: $\mathcal{O}(\log \log |X|)$ bits in expectation.
Updates: insert ✓, merge ✓

Intuition

$|X|$ elements have ≥ 0 leading zeroes
$\approx |X|/2$ elements have ≥ 1 leading zero
$\approx |X|/4$ elements have ≥ 2 leading zeroes
...
$\approx |X|/2^k$ elements have $\geq k$ leading zeroes
$\hookrightarrow$ highest number of leading zeroes is $\approx \log |X|$.

Example

$x \in X$	$h(x)$ in binary	leading zeroes
E	**00**101...	2
X	10111...	0
A	**000**10...	**3**
M	10110...	0
P	10110...	0
L	**0**1011...	1
E	**00**101...	2

largest number of leading zeroes is $S(X) = 3$.
$\Rightarrow$ estimate $|X| \approx 2^3 = 8$.

Remark: See HyperLogLog Sketch

- $2^{S(X)} \in [|X|/4, 8|X|]$ with probability $> 70\%$
- for more precise estimates of $|X|$
  - use $k \in \mathbb{N}$ copies of the sketch
  - each element contributes to one sketch chosen by $h_0 : U \to [k]$
  - use harmonic mean of estimates
  - multiply with k and magic constant

Set Sketch for Membership Estimation // much like a Bloom Filter

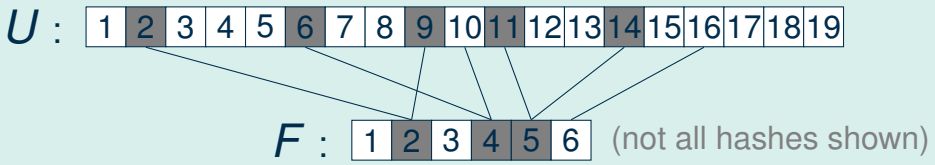
Sketch Definition and Purpose

Parameters: $u = |U|$, $n \leq u$ and $\epsilon > 0$.
Let $F = [n/\epsilon]$ and $\mathrm{f} : U \to F$ be a hash function.

$$S : \begin{cases} 2^U & \to 2^F \\ X & \mapsto \{\mathrm{f}(x) \mid x \in X\} \end{cases}$$ // set of "fingerprints"

Goal: Estimate $[x \in X]$ as $[\mathrm{f}(x) \in S(X)]$.
Updates: insert ✓, merge ✓

Example



6 is true positive, 10 is false positive, 16 is true negative

Set Sketch for Membership Estimation // much like a Bloom Filter

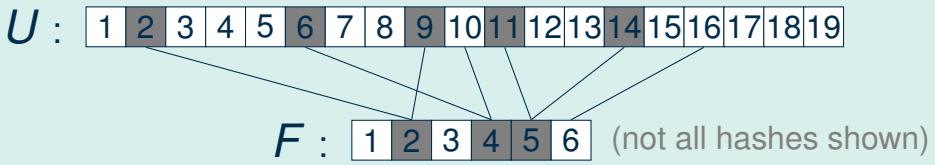
Sketch Definition and Purpose

Parameters: $u = |U|$, $n \leq u$ and $\epsilon > 0$.
Let $F = [n/\epsilon]$ and $\mathrm{f} : U \to F$ be a hash function.

$$S : \begin{cases} 2^U & \to 2^F \\ X & \mapsto \{\mathrm{f}(x) \mid x \in X\} \end{cases}$$ // set of "fingerprints"

Goal: Estimate $[x \in X]$ as $[\mathrm{f}(x) \in S(X)]$.
Updates: insert ✓, merge ✓

Example



6 is true positive, 10 is false positive, 16 is true negative

Properties

- no false negatives: $x \in X \Rightarrow h(x) \in S(X)$
- if $|X| \leq n$, then few false positives:

$$y \notin X \Rightarrow \Pr[f(y) \in S(X)] \leq \sum_{x \in X} \Pr[f(y) = f(x)] = \frac{|X|}{|F|} \leq \frac{n}{n/\epsilon} = \epsilon.$$

- significant space savings if $u \gg n = |X|$ and ϵ not too small:
  - $\mathrm{space}(X) = \log_2 \binom{u}{n} \approx \log_2 \left(\frac{ue}{n}\right)^n = n \log_2(ue/n)$
  - $\mathrm{space}(S(X)) = \log_2 \binom{|F|}{n} = \log_2 \binom{n/\epsilon}{n} \leq \log_2 \left(\frac{en/\epsilon}{n}\right)^n = n \log_2(e/\epsilon)$.
  - Example: $u = 2^{64}$, $n = 2^{32}$, $\epsilon = 2^{-8}$.
    $\rightsquigarrow$ from $\geq 33.4n$ bits to $9.4n$ bits.

Set Sketch for Membership Estimation // much like a Bloom Filter

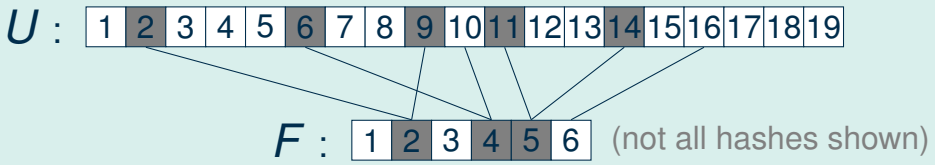
Sketch Definition and Purpose

Parameters: $u = |U|$, $n \leq u$ and $\epsilon > 0$.
Let $F = [n/\epsilon]$ and $f : U \to F$ be a hash function.

$$S : \begin{cases} 2^U & \to 2^F \\ X & \mapsto \{f(x) \mid x \in X\} \end{cases}$$ // set of "fingerprints"

Goal: Estimate $[x \in X]$ as $[f(x) \in S(X)]$.
Updates: insert ✓, merge ✓

Example



6 is true positive, 10 is false positive, 16 is true negative

Properties

- no false negatives: $x \in X \Rightarrow h(x) \in S(X)$
- if $|X| \leq n$, then few false positives:

$$y \notin X \Rightarrow \Pr[f(y) \in S(X)] \leq \sum_{x \in X} \Pr[f(y) = f(x)] = \frac{|X|}{|F|} \leq \frac{n}{n/\epsilon} = \epsilon.$$

- significant space savings if $u \gg n = |X|$ and ϵ not too small:
  - $\text{space}(X) = \log_2 \binom{u}{n} \approx \log_2 \left(\frac{ue}{n}\right)^n = n \log_2(ue/n)$
  - $\text{space}(S(X)) = \log_2 \binom{|F|}{n} = \log_2 \binom{n/\epsilon}{n} \leq \log_2 \left(\frac{en/\epsilon}{n}\right)^n = n \log_2(e/\epsilon)$.
  - Example: $u = 2^{64}$, $n = 2^{32}$, $\epsilon = 2^{-8}$.
    $\rightsquigarrow$ from $\geq 33.4n$ bits to $9.4n$ bits.

Remark: See Quotient Filter

- Conceptually: Compressed Single-Shot Bloom Filter.
- Supports deletions if $\{f(x) \mid x \in X\}$ stored as multiset.

Interlude: Evaluation

https://onlineumfrage.kit.edu/evasys/online.php?p=L52WE

Sketches of Sets, Vectors and Graphs

1. Set Sketches...

... for Cardinality Estimation
... for Membership Estimation

2. Vector Sketches and the Turnstile Model

... for $\|x\|_2$-Norm Estimation
... for Heavy Hitter Detection

3. Set Sketches for Sparse Recovery

Warm-Up: 1-Sparse Recovery
n-Sparse Recovery
Getting Any Element
Application: Graph Sketch for Spanning Tree Computation

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Targeted Queries

- vector norm $\|x\|_p$ for $p \in \{0\} \cup (1, \infty]$
  // technically not a norm for $p = 0$
- heavy hitters: $\{i \in [n] \mid x_i$ is "large"$\}$
- sparse recovery: x whenever $\|x\|_0$ is "small"

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Targeted Queries

- vector norm $\|x\|_p$ for $p \in \{0\} \cup (1, \infty]$
  // technically not a norm for $p = 0$
- heavy hitters: $\{i \in [n] \mid x_i \text{ is “large”}\}$
- sparse recovery: x whenever $\|x\|_0$ is “small”

Examples for

- Sketch for approximating $\|x\|_0$ in cache register model

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Targeted Queries

- vector norm $\|x\|_p$ for $p \in \{0\} \cup (1, \infty]$
  // technically not a norm for $p = 0$
- heavy hitters: $\{i \in [n] \mid x_i \text{ is “large”}\}$
- sparse recovery: x whenever $\|x\|_0$ is “small”

Examples for

- Sketch for approximating $\|x\|_0$ in cache register model
  ↪ count distinct problem, see HyperLogLog

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Targeted Queries

- vector norm $\|x\|_p$ for $p \in \{0\} \cup (1, \infty]$
  // technically not a norm for $p = 0$
- heavy hitters: $\{i \in [n] \mid x_i \text{ is “large”}\}$
- sparse recovery: x whenever $\|x\|_0$ is “small”

Examples for

- Sketch for approximating $\|x\|_0$ in cache register model
  ↪ count distinct problem, see HyperLogLog
- Sketch for $\|x\|_1$

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Targeted Queries

- vector norm $\|x\|_p$ for $p \in \{0\} \cup (1, \infty]$
  // technically not a norm for $p = 0$
- heavy hitters: $\{i \in [n] \mid x_i \text{ is “large”}\}$
- sparse recovery: x whenever $\|x\|_0$ is “small”

Examples for

- Sketch for approximating $\|x\|_0$ in cache register model
  ↪ count distinct problem, see HyperLogLog
- Sketch for $\|x\|_1$
  - in strict turnstile: Trivial in $\mathcal{O}(\log\|x\|_1)$ space since $\|x\|_1 = \sum_{i\in[n]} |x_i| \overset{x\geq 0}{=} \sum_{i\in[n]} x_i$. // just use one counter

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Targeted Queries

- vector norm $\|x\|_p$ for $p \in \{0\} \cup (1, \infty]$
  // technically not a norm for $p = 0$
- heavy hitters: $\{i \in [n] \mid x_i \text{ is “large”}\}$
- sparse recovery: x whenever $\|x\|_0$ is “small”

Examples for

- Sketch for approximating $\|x\|_0$ in cache register model
  ↪ count distinct problem, see HyperLogLog
- Sketch for $\|x\|_1$
  - in strict turnstile: Trivial in $\mathcal{O}(\log\|x\|_1)$ space since $\|x\|_1 = \sum_{i\in[n]} |x_i| \overset{x\geq 0}{=} \sum_{i\in[n]} x_i$. // just use one counter
  - in general turnstile model: tricky // something with Cauchy distributions, not in this lecture

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Targeted Queries

- vector norm $\|x\|_p$ for $p \in \{0\} \cup (1, \infty]$
  // technically not a norm for $p = 0$
- heavy hitters: $\{i \in [n] \mid x_i \text{ is “large”}\}$
- sparse recovery: x whenever $\|x\|_0$ is “small”

Examples for

- Sketch for approximating $\|x\|_0$ in cache register model
  ↪ count distinct problem, see HyperLogLog
- Sketch for $\|x\|_1$
  - in strict turnstile: Trivial in $\mathcal{O}(\log \|x\|_1)$ space since $\|x\|_1 = \sum_{i \in [n]} |x_i| \overset{x \geq 0}{=} \sum_{i \in [n]} x_i$. // just use one counter
  - in general turnstile model: tricky // something with Cauchy distributions, not in this lecture
  - in cache register model: approximate counting
    Morris’s algorithm counts m events in space $\widetilde{O}(\log \log m)$, see probability and computing lecture

Vector Sketches and the Turnstile Model // turnstile = Drehkreuz

Setting

Sketch $x \in \mathbb{Z}^n$, initially $x = \vec{0}$, supporting updates $\langle i, \Delta \rangle$ that add Δ to x_i.

- Cash Register Model: $\Delta \in \mathbb{N}$.
- Strict Turnstile Model: $\Delta \in \mathbb{Z}$, but $x \geq 0$ always.
- Turnstile Model: no restriction // sometimes $x \in \mathbb{Z}_k^n$.

Targeted Queries

- vector norm $\|x\|_p$ for $p \in \{0\} \cup (1, \infty]$
  // technically not a norm for $p = 0$
- heavy hitters: $\{i \in [n] \mid x_i \text{ is “large”}\}$
- sparse recovery: x whenever $\|x\|_0$ is “small”

Examples for

- Sketch for approximating $\|x\|_0$ in cache register model
  ↪ count distinct problem, see HyperLogLog
- Sketch for $\|x\|_1$
  - in strict turnstile: Trivial in $\mathcal{O}(\log \|x\|_1)$ space since $\|x\|_1 = \sum_{i \in [n]} |x_i| \overset{x \geq 0}{=} \sum_{i \in [n]} x_i$. // just use one counter
  - in general turnstile model: tricky // something with Cauchy distributions, not in this lecture
  - in cache register model: approximate counting
    Morris's algorithm counts m events in space $\widetilde{O}(\log \log m)$, see probability and computing lecture
- Next: sketch for approximating $\|x\|_2$ in turnstile model.

Sketch for $\|x\|_2^2$ and $\langle x, y \rangle$ in turnstile model [AMS1999]

Sketch Definition

Let $h : [n] \to \{-1, 1\}$ be a hash function.

$$S : \begin{cases} \mathbb{Z}^n & \to \mathbb{Z} \\ (x_1, \ldots, x_n) & \mapsto \sum_{i=1}^{n} h(i) \cdot x_i \end{cases}$$

Goal: estimate $\|x\|_2^2$ as $S(x)^2$ and
estimate $\langle x, y \rangle$ as $S(x) \cdot S(y)$

Updates: general turnstile ✓
merging: $S(x + y) = S(x) + S(y)$ ✓

Sketch for $\|x\|_2^2$ and $\langle x, y \rangle$ in turnstile model [AMS1999]

Sketch Definition

Let $h : [n] \to \{-1, 1\}$ be a hash function.

$$S : \begin{cases} \mathbb{Z}^n & \to \mathbb{Z} \\ (x_1, \ldots, x_n) & \mapsto \sum_{i=1}^{n} h(i) \cdot x_i \end{cases}$$

Goal: estimate $\|x\|_2^2$ as $S(x)^2$ and
estimate $\langle x, y \rangle$ as $S(x) \cdot S(y)$

Updates: general turnstile ✓
merging: $S(x + y) = S(x) + S(y)$ ✓

Lemma

For any $x, y \in \mathbb{Z}^n$ we have

1. $\mathbb{E}[S(x) \cdot S(y)] = \langle x, y \rangle$
2. $\mathrm{Var}(S(x) \cdot S(y)) = \mathcal{O}(\|x\|_2^2 \|y\|_2^2)$ // proof omitted
3. $\mathbb{E}[S(x)^2] = \|x\|_2^2$
4. $\mathrm{Var}[S(x)^2] = \|x\|_2^4$

Sketch for $\|x\|_2^2$ and $\langle x, y\rangle$ in turnstile model [AMS1999]

Sketch Definition

Let $h : [n] \to \{-1, 1\}$ be a hash function.

$$S : \begin{cases} \mathbb{Z}^n & \to \mathbb{Z} \\ (x_1, \ldots, x_n) & \mapsto \sum_{i=1}^{n} h(i) \cdot x_i \end{cases}$$

Goal: estimate $\|x\|_2^2$ as $S(x)^2$ and
estimate $\langle x, y\rangle$ as $S(x) \cdot S(y)$

Updates: general turnstile ✓
merging: $S(x + y) = S(x) + S(y)$ ✓

Lemma

For any $x, y \in \mathbb{Z}^n$ we have

1. $\mathbb{E}[S(x) \cdot S(y)] = \langle x, y\rangle$
2. $\mathrm{Var}(S(x) \cdot S(y)) = \mathcal{O}(\|x\|_2^2 \|y\|_2^2)$ // proof omitted
3. $\mathbb{E}[S(x)^2] = \|x\|_2^2$
4. $\mathrm{Var}[S(x)^2] = \|x\|_2^4$

Proof of (1)

$$\begin{aligned}
\mathbb{E}[S(x)S(y)] &= \mathbb{E}\Big[\big(\sum_{i=1}^{n} h(i)x_i\big) \cdot \big(\sum_{j=1}^{n} h(j)y_j\big)\Big] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_j \mathbb{E}[h(i)h(j)] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_j [i = j] \\
&= \sum_{i=1}^{n} x_i y_i = \langle x, y\rangle.
\end{aligned}$$

Sketch for $\|x\|_2^2$ and $\langle x, y\rangle$ in turnstile model [AMS1999]

Sketch Definition

Let $h : [n] \to \{-1, 1\}$ be a hash function.

$$S : \begin{cases} \mathbb{Z}^n & \to \mathbb{Z} \\ (x_1, \ldots, x_n) & \mapsto \sum_{i=1}^{n} h(i) \cdot x_i \end{cases}$$

Goal: estimate $\|x\|_2^2$ as $S(x)^2$ and
estimate $\langle x, y\rangle$ as $S(x) \cdot S(y)$

Updates: general turnstile ✓
merging: $S(x + y) = S(x) + S(y)$ ✓

Lemma

For any $x, y \in \mathbb{Z}^n$ we have

1. $\mathbb{E}[S(x) \cdot S(y)] = \langle x, y\rangle$
2. $\mathrm{Var}(S(x) \cdot S(y)) = \mathcal{O}(\|x\|_2^2 \|y\|_2^2)$ // proof omitted
3. $\mathbb{E}[S(x)^2] = \|x\|_2^2$
4. $\mathrm{Var}[S(x)^2] = \|x\|_2^4$

Is this useful?

- S yields unbiased estimator for $\langle x, y\rangle$ and $\|x\|_2^2$ in $\mathcal{O}(1)$ space ✓
- Standard technique: Average t independent estimates
  $\Rightarrow$ variance drops by factor t
  - $\Rightarrow$ Estimate of $\|x\|_2^2$ with standard deviation $\frac{1}{\sqrt{t}}\|x\|_2^2$ // good approximation
  - $\Rightarrow$ Estimate of $\langle x, y\rangle$ with standard deviation $\frac{1}{\sqrt{t}}\|x\|_2\|y\|_2$ // may not be good

Sketch for $\|x\|_2^2$ and $\langle x, y\rangle$ in turnstile model [AMS1999]

Sketch Definition

Let $h : [n] \to \{-1, 1\}$ be a hash function.

$$S : \begin{cases} \mathbb{Z}^n & \to \mathbb{Z} \\ (x_1, \ldots, x_n) & \mapsto \sum_{i=1}^{n} h(i) \cdot x_i \end{cases}$$

Goal: estimate $\|x\|_2^2$ as $S(x)^2$ and estimate $\langle x, y\rangle$ as $S(x) \cdot S(y)$

Updates: general turnstile ✓
merging: $S(x + y) = S(x) + S(y)$ ✓

Lemma

For any $x, y \in \mathbb{Z}^n$ we have

1. $\mathbb{E}[S(x) \cdot S(y)] = \langle x, y\rangle$
2. $\mathrm{Var}(S(x) \cdot S(y)) = \mathcal{O}(\|x\|_2^2 \|y\|_2^2)$ // proof omitted
3. $\mathbb{E}[S(x)^2] = \|x\|_2^2$
4. $\mathrm{Var}[S(x)^2] = \|x\|_2^4$

Is this useful?

- S yields unbiased estimator for $\langle x, y\rangle$ and $\|x\|_2^2$ in $\mathcal{O}(1)$ space ✓
- Standard technique: Average t independent estimates
  $\Rightarrow$ variance drops by factor t
  - $\Rightarrow$ Estimate of $\|x\|_2^2$ with standard deviation $\frac{1}{\sqrt{t}}\|x\|_2^2$ // good approximation
  - $\Rightarrow$ Estimate of $\langle x, y\rangle$ with standard deviation $\frac{1}{\sqrt{t}}\|x\|_2\|y\|_2$ // may not be good

Application

- Estimate size of data base joins
  // self-join in case of $\|x\|_2^2$
- Allows optimising order in which complex queries are evaluated.

Heavy Hitter Detection & Count Min Sketch

Theorem: Count Min Sketch

For any $t \in \mathbb{N}$ and $\epsilon > 0$ there exists a sketch for $x \in \mathbb{N}_0^n$ that can estimate x_i as $\hat{x}_i$ such that

- $\hat{x}_i \geq x_i$ // deterministically
- $\Pr[\hat{x}_i \leq x_i + \epsilon\|x\|_1] \geq 1 - 2^{-t}$
- the sketch uses $\mathcal{O}(t/\epsilon)$ space

Application: Heavy Hitter Detection

- call $i \in [n]$ *heavy hitter* $x_i \geq 2\epsilon\|x\|_1$
  // e.g. network flow i makes up for more than $2\epsilon = 5\%$ of packets.
- call $i \in [n]$ *light* $x_i \leq \epsilon\|x\|_1$

Classify i as heavy if $\hat{x}_i \geq 2\epsilon\|x\|_1$ and as *light* otherwise.

- i is heavy hitter $\Rightarrow$ i is classified correctly
- i is light $\Rightarrow \Pr[i \text{ classified correctly }] \geq 1 - 2^{-t}$.
- *no guarantee for intermediate elements*

Sketch Definition for $t = 1$

Let $k = 2/\epsilon$ and $h : [n] \to [k]$ a hash function.

$$S : \begin{cases} \mathbb{N}_0^n & \to \mathbb{N}_0^k \\ (x_1, \ldots, x_n) & \mapsto \sum_{i=1}^n \vec{e}_{h(i)} \cdot x_i \end{cases}$$

Estimate: $\hat{x}_i := S(x)_{h(i)}$
Updates: strict turnstile ✓ merging ✓

Heavy Hitter Detection & Count Min Sketch

Theorem: Count Min Sketch

For any $t \in \mathbb{N}$ and $\epsilon > 0$ there exists a sketch for $x \in \mathbb{N}_0^n$ that can estimate x_i as $\hat{x}_i$ such that

- $\hat{x}_i \geq x_i$ // deterministically
- $\Pr[\hat{x}_i \leq x_i + \epsilon\|x\|_1] \geq 1 - 2^{-t}$
- the sketch uses $\mathcal{O}(t/\epsilon)$ space

Application: Heavy Hitter Detection

- call $i \in [n]$ *heavy hitter* $x_i \geq 2\epsilon\|x\|_1$
  // e.g. network flow i makes up for more than $2\epsilon = 5\%$ of packets.
- call $i \in [n]$ *light* $x_i \leq \epsilon\|x\|_1$

Classify i as heavy if $\hat{x}_i \geq 2\epsilon\|x\|_1$ and as *light* otherwise.

- i is heavy hitter $\Rightarrow$ i is classified correctly
- i is light $\Rightarrow \Pr[i \text{ classified correctly }] \geq 1 - 2^{-t}$.
- *no guarantee for intermediate elements*

Sketch Definition for $t = 1$

Let $k = 2/\epsilon$ and $h : [n] \to [k]$ a hash function.

$$S : \begin{cases} \mathbb{N}_0^n & \to \mathbb{N}_0^k \\ (x_1, \ldots, x_n) & \mapsto \sum_{i=1}^n \vec{e}_{h(i)} \cdot x_i \end{cases}$$

Estimate: $\hat{x}_i := S(x)_{h(i)}$
Updates: strict turnstile ✓ merging ✓

Properties for $t = 1$

- $\hat{x}_i \geq x_i$. ✓
- $\Pr[\hat{x}_i \geq x_i + \epsilon\|x\|_1] = \Pr[S(X)_{h(i)} - x_i \geq \frac{2\|x\|_1}{k}]$
  $\leq \frac{\mathbb{E}[S(x)_{h(i)} - x_i]}{2\|x\|_1/k} \leq \frac{\|x\|_1/k}{2\|x\|_1/k} = \frac{1}{2}$.

Heavy Hitter Detection & Count Min Sketch

Theorem: Count Min Sketch

For any $t \in \mathbb{N}$ and $\epsilon > 0$ there exists a sketch for $x \in \mathbb{N}_0^n$ that can estimate x_i as $\hat{x}_i$ such that

- $\hat{x}_i \geq x_i$ // deterministically
- $\Pr[\hat{x}_i \leq x_i + \epsilon\|x\|_1] \geq 1 - 2^{-t}$
- the sketch uses $\mathcal{O}(t/\epsilon)$ space

Application: Heavy Hitter Detection

- call $i \in [n]$ *heavy hitter* $x_i \geq 2\epsilon\|x\|_1$
  // e.g. network flow i makes up for more than $2\epsilon = 5\%$ of packets.
- call $i \in [n]$ *light* $x_i \leq \epsilon\|x\|_1$

Classify i as heavy if $\hat{x}_i \geq 2\epsilon\|x\|_1$ and as *light* otherwise.

- i is heavy hitter $\Rightarrow$ i is classified correctly
- i is light $\Rightarrow \Pr[i \text{ classified correctly }] \geq 1 - 2^{-t}$.
- *no guarantee for intermediate elements*

Sketch Definition for $t = 1$

Let $k = 2/\epsilon$ and $h : [n] \to [k]$ a hash function.

$$S : \begin{cases} \mathbb{N}_0^n & \to \mathbb{N}_0^k \\ (x_1, \ldots, x_n) & \mapsto \sum_{i=1}^n \vec{e}_{h(i)} \cdot x_i \end{cases}$$

Estimate: $\hat{x}_i := S(x)_{h(i)}$
Updates: strict turnstile ✓ merging ✓

Properties for $t = 1$

- $\hat{x}_i \geq x_i$. ✓
- $\Pr[\hat{x}_i \geq x_i + \epsilon\|x\|_1] = \Pr[S(X)_{h(i)} - x_i \geq \frac{2\|x\|_1}{k}]$
  $\leq \frac{\mathbb{E}[S(x)_{h(i)} - x_i]}{2\|x\|_1/k} \leq \frac{\|x\|_1/k}{2\|x\|_1/k} = \frac{1}{2}$.

Sketch Definition for $t > 1$

- instantiate t copies of the $(t = 1)$-sketch
- $\hat{x}_i := \min_{j \in [t]} S^{(j)}(x)_i$.

Sketches of Sets, Vectors and Graphs

1. Set Sketches...

... for Cardinality Estimation
... for Membership Estimation

2. Vector Sketches and the Turnstile Model

... for $\|x\|_2$-Norm Estimation
... for Heavy Hitter Detection

3. Set Sketches for Sparse Recovery

Warm-Up: 1-Sparse Recovery
n-Sparse Recovery
Getting Any Element
Application: Graph Sketch for Spanning Tree Computation

The Benefits of *Linear* Sketching

Note: The Previous two Sketches are Linear

A vector sketch is linear if

$$\forall \text{vectors } x, y : \forall \text{scalars } a, b : S(a \cdot x + b \cdot y) = a \cdot S(x) + b \cdot S(y)$$

- sketch for $\|x\|_2^2$:

$$S(x) = \sum_{i=1}^{n} h(i) \cdot x_i = \underbrace{(-1\ 1\ 1\ -1 \ldots\ -1)}_{\text{random vector in } \{0,1\}^n} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

- count min sketch ($t = 1$):

$$S(x) = \sum_{i=1}^{n} \vec{e}_{h(i)} \cdot x_i = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & \ldots & 0 \\ 0 & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 0 & 0 & \ldots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 1 & 0 & 1 & \ldots & 0 \end{pmatrix}}_{\text{one “1” per column placed randomly}} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

The Benefits of *Linear* Sketching

Note: The Previous two Sketches are Linear

A vector sketch is linear if

$$\forall \text{vectors } x, y : \forall \text{scalars } a, b : S(a \cdot x + b \cdot y) = a \cdot S(x) + b \cdot S(y)$$

- sketch for $\|x\|_2^2$:

$$S(x) = \sum_{i=1}^{n} h(i) \cdot x_i = \underbrace{(-1\ 1\ 1\ -1 \ldots\ -1)}_{\text{random vector in } \{0,1\}^n} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

- count min sketch ($t = 1$):

$$S(x) = \sum_{i=1}^{n} \vec{e}_{h(i)} \cdot x_i = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & \ldots & 0 \\ 0 & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 0 & 0 & \ldots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 1 & 0 & 1 & \ldots & 0 \end{pmatrix}}_{\text{one “1” per column placed randomly}} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Benefits

- linearity implies mergeability
  $\Rightarrow$ distributed computation possible
- linearity implies deletions:
  $S(x) + S(-x) = 0$
- sketch of the difference is the difference of the sketches:

$$S(x) + S(-x') = S(x - x').$$

Motivation: Sparse Recovery

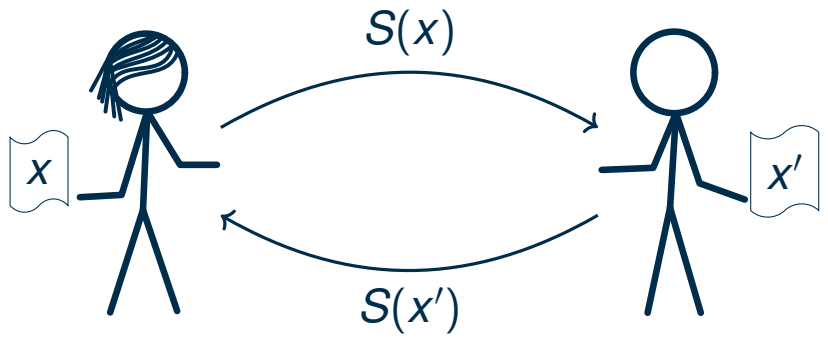
Sparse Recovery

Implement a “recover” algorithm such that recover($S(x)$) $= x$ whenever x is “sparse”. // e.g. $\|x\|_0 < \epsilon$

Lemma Application: Set Reconciliation

Assume Alice and Bob hold “similar” vectors x and x', meaning $x - x'$ is sparse.
If S is a linear sketch for sparse recovery, then Alice can compute x' after Bob sends $S(x')$.

Proof



Motivation: Sparse Recovery

Sparse Recovery

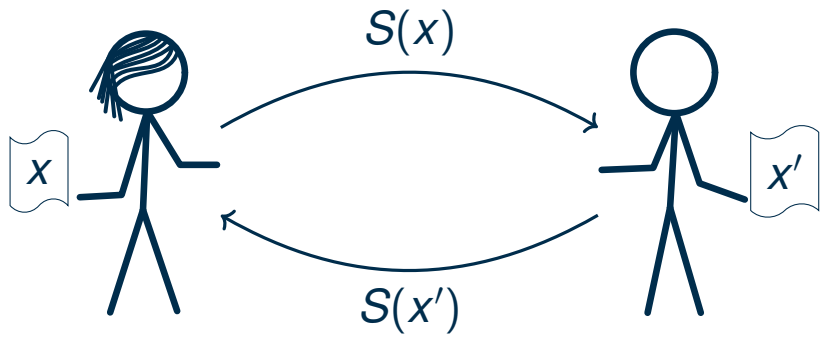
Implement a “recover” algorithm such that $\text{recover}(S(x)) = x$ whenever x is “sparse”. // e.g. $\|x\|_0 < \epsilon$

Lemma Application: Set Reconciliation

Assume Alice and Bob hold “similar” vectors x and x', meaning $x - x'$ is sparse.
If S is a linear sketch for sparse recovery, then Alice can compute x' after Bob sends $S(x')$.

Proof

- Alice computes $S(x)$ and receives $S(x')$ from Bob
- Alice computes $S(x' - x) = S(x') - S(x)$ // linearity
- Alice computes $x' - x = \text{recover}(S(x' - x))$ // sparsity
- Alice computes $x' = (x' - x) + x$



KIT

Warm-Up: 1-Sparse Set Recovery from $O(1)$ words

Starting Point

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: incorrect result

Warm-Up: 1-Sparse Set Recovery from $O(1)$ words

Starting Point

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: incorrect result

Sketch Definition and Recovery Algorithm

Let $U = \{1, \dots, 2^w - 1\}$. We sketch subsets $X \subseteq U$.

$$S : \begin{cases} 2^U & \to \{0,1\}^w \\ X & \mapsto \bigoplus_{x \in X} x \end{cases}$$

Algorithm recover(sum $= S(X)$)**:**

if sum $= 0$ **then return** $\emptyset$ // note $0 \notin U$

else return $\{\text{sum}\}$ // singleton

Warm-Up: 1-Sparse Set Recovery from $O(1)$ words

Starting Point

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: incorrect result

Is this a Linear Sketch?

Sketch Definition and Recovery Algorithm

Let $U = \{1, \ldots, 2^w - 1\}$. We sketch subsets $X \subseteq U$.

$$S : \begin{cases} 2^U & \to \{0,1\}^w \\ X & \mapsto \bigoplus_{x \in X} x \end{cases}$$

Algorithm recover(sum $= S(X)$)**:**

if sum $= 0$ **then return** $\emptyset$ // note $0 \notin U$

else return $\{\text{sum}\}$ // singleton

Warm-Up: 1-Sparse Set Recovery from $O(1)$ words

Starting Point

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: incorrect result

Is this a Linear Sketch?

Yes with symmetric difference on sets: $S(X \triangle Y) = S(X) \oplus S(Y)$.

Sketch Definition and Recovery Algorithm

Let $U = \{1, \ldots, 2^w - 1\}$. We sketch subsets $X \subseteq U$.

$$S : \begin{cases} 2^U & \to \{0,1\}^w \\ X & \mapsto \bigoplus_{x \in X} x \end{cases}$$

Algorithm recover(sum $= S(X)$)**:**

if sum $= 0$ **then return** $\emptyset$ // note $0 \notin U$

else return $\{\text{sum}\}$ // singleton

Warm-Up: 1-Sparse Set Recovery from $O(1)$ words

Starting Point

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: incorrect result

Is this a Linear Sketch?

Yes with symmetric difference on sets: $S(X \triangle Y) = S(X) \oplus S(Y)$.

Step 2:

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: $\Pr[\text{recover}(S(X)) = \text{NOT-SPARSE}] \geq 1 - 2^{-w}$

Sketch Definition and Recovery Algorithm

Let $U = \{1, \ldots, 2^w - 1\}$. We sketch subsets $X \subseteq U$.

$$S : \begin{cases} 2^U & \to \{0,1\}^w \\ X & \mapsto \bigoplus_{x \in X} x \end{cases}$$

Algorithm recover(sum $= S(X)$)**:**

if sum $= 0$ **then return** $\emptyset$ // note $0 \notin U$

else return $\{\text{sum}\}$ // singleton

Warm-Up: 1-Sparse Set Recovery from $O(1)$ words

Starting Point

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: incorrect result

Is this a Linear Sketch?

Yes with symmetric difference on sets: $S(X \triangle Y) = S(X) \oplus S(Y)$.

Step 2: Checksum for Robustness

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: $\Pr[\text{recover}(S(X)) = \text{NOT-SPARSE}] \geq 1 - 2^{-w}$

Sketch Definition and Recovery Algorithm

Let $U = \{1, \ldots, 2^w - 1\}$ and $h : U \to \{0,1\}^w$ a hash function.
We sketch subsets $X \subseteq U$.

$$S : \begin{cases} 2^U & \to (\{0,1\}^w)^2 \\ X & \mapsto (\underbrace{\bigoplus_{x \in X} x}_{\text{sum}}, \underbrace{\bigoplus_{x \in X} h(x)}_{\text{hashSum}}) \end{cases}$$

Algorithm recover$((\text{sum}, \text{hashSum}) = S(X))$**:**
  if $(\text{sum}, \text{hashSum}) = (0,0)$ **then return** $\emptyset$
  if $\text{sum} = 0$ *or* $h(\text{sum}) \neq \text{hashSum}$ **then**
    return NOT-SPARSE
  else return $\{\text{sum}\}$ // singleton

Warm-Up: 1-Sparse Set Recovery from $O(1)$ words

Starting Point

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: incorrect result

Is this a Linear Sketch?

Yes with symmetric difference on sets: $S(X \triangle Y) = S(X) \oplus S(Y)$.

Step 2: Checksum for Robustness

- if $|X| \leq 1$: correct recovery guaranteed
- if $|X| > 1$: $\Pr[\text{recover}(S(X)) = \text{NOT-SPARSE}] \geq 1 - 2^{-w}$

Step 3: Support for Multisets

- if $\{x \in U \mid M(x) \neq 0\} \leq 1$: correct recovery guaranteed
- otherwise: $\Pr[\text{recover}(S(M)) = \text{NOT-SPARSE}] \geq 1 - 2^{-w}$

Sketch Definition and Recovery Algorithm

Let p be prime, $U = \mathbb{F}_p^* = \{1, \dots, p-1\}$ and $h : U \to \mathbb{F}_p$ be a hash function. Consider multisets M of U with multiplicities in $\mathbb{F}_p$, formalised as functions $M : \mathbb{F}_p^* \to \mathbb{F}_p$.

$$S : \begin{cases} \mathbb{F}_p^{\mathbb{F}_p^*} & \to \mathbb{F}_p^3 \\ M & \mapsto (\underbrace{\sum_{i \in U} M(i) \cdot i}_{\text{sum}}, \underbrace{\sum_{i \in U} M(i)}_{\text{count}}, \underbrace{\sum_{i \in U} M(i) \cdot h(i)}_{\text{hashSum}}) \end{cases}$$

Algorithm recover((sum, count, hashSum) $= S(M)$)**:**

if (sum, count, hashSum) $= (0, 0, 0)$ **then return** $\emptyset$

if sum $= 0$ *or* count $= 0$ *or* $h(\frac{\text{sum}}{\text{count}}) \neq \frac{\text{hashSum}}{\text{count}}$ **then**

return NOT-SPARSE

else return "count copies of (sum/count)"

Example $M = \{2, 2, 4\}$ // $M(2) = 2$ and $M(4) = 1$

$S(M) = (\text{sum}, \text{count}, \text{hashSum}) = (2 \cdot 2 + 1 \cdot 4, 2 + 1, 2 \cdot h(2) + 1 \cdot h(4))$

KIT

Teaser: n-Sparse Recovery from 1-Sparse Recovery

"Invertible Bloom Lookup Table"

- uses array $A[1..n']$ of $n' = 1.23k$ sketches for 1-sparse set recovery
- uses hash functions $h_1, h_2, h_3 : U \to [n']$
- insert($x \in U$): adds x to $A[h_1(x)]$, $A[h_2(x)]$ and $A[h_3(x)]$
- delete($x \in U$): deletes x from $A[h_1(x)]$, $A[h_2(x)]$ and $A[h_3(x)]$

Algorithm recover($A[1..n']$)**:**

```
X ← ∅
while ∃i ∈ [n'] : A[i] reports singleton set {x} do
    X ← X ∪{x}
    delete(x)
if ∀i ∈ [n'] : A[i] reports ∅ then
    return X
else
    return NOT-SPARSE
```

Theorem

- if $|X| \leq n$: correct recovery with high probability
- if $|X| > n$: returns NOT-SPARSE with high probability

Remarks

- proof in probability and computing lecture
- see attached file 📎 for illustrations of a variant without checksums

KIT

1. Set Sketches...
   - ... for Cardinality Estimation
   - ... for Membership Estimation

2. Vector Sketches and the Turnstile Model
   - ... for $\|x\|_2$-Norm Estimation
   - ... for Heavy Hitter Detection

3. Set Sketches for Sparse Recovery
   - Warm-Up: 1-Sparse Recovery
   - n-Sparse Recovery
   - Getting Any Element
   - Application: Graph Sketch for Spanning Tree Computation

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then $\text{getAny}(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then $\text{getAny}(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \textsc{empty}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\textsc{fail}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \textsc{fail}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

if $\forall i \in \{0,\dots,k\}$: recover$(A[i]) = \emptyset$ **then return** $\emptyset$
if $\exists i \in \{0,\dots,k\}$: recover$(A[i])$ *is singleton* $\{x\}$ **then**
  return x
return FAIL

Proof (Analysis of the failure event)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

if $\forall i \in \{0,\dots,k\}$: recover$(A[i]) = \emptyset$ **then return** $\emptyset$
if $\exists i \in \{0,\dots,k\}$: recover$(A[i])$ *is singleton* $\{x\}$ **then**
  return x
return FAIL

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) =$
$h(x_2) =$
$h(x_3) =$
$h(x_4) =$
$h(x_5) =$

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

if $\forall i \in \{0,\dots,k\}$: recover$(A[i]) = \emptyset$ **then return** $\emptyset$
if $\exists i \in \{0,\dots,k\}$: recover$(A[i])$ *is singleton* $\{x\}$ **then**
  return x
return FAIL

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = \ 0$
$h(x_2) =$
$h(x_3) =$
$h(x_4) =$
$h(x_5) =$

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then $\text{getAny}(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then $\text{getAny}(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A):

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) =$ 0
$h(x_2) =$ 1 (added to $A[0]$)
$h(x_3) =$
$h(x_4) =$
$h(x_5) =$

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \textsc{empty}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\textsc{fail}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \textsc{fail}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

if $\forall i \in \{0,\dots,k\}$: recover$(A[i]) = \emptyset$ **then return** $\emptyset$
if $\exists i \in \{0,\dots,k\}$: recover$(A[i])$ *is singleton* $\{x\}$ **then**
  return x
return FAIL

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 0$
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 0$
$h(x_4) =$
$h(x_5) =$

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 0$
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 0$
$h(x_4) = 0$
$h(x_5) =$

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \textsc{empty}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\textsc{fail}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \textsc{fail}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 0$
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 0$
$h(x_4) = 0$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

if $\forall i \in \{0,\dots,k\}$: recover$(A[i]) = \emptyset$ **then return** $\emptyset$
if $\exists i \in \{0,\dots,k\}$: recover$(A[i])$ *is singleton* $\{x\}$ **then**
  return x
return FAIL

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 00$
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 0$
$h(x_4) = 0$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 00$
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 00$
$h(x_4) = 0$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 00$
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 00$
$h(x_4) = 00$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

if $\forall i \in \{0,\dots,k\}$: recover$(A[i]) = \emptyset$ **then return** $\emptyset$
if $\exists i \in \{0,\dots,k\}$: recover$(A[i])$ *is singleton* $\{x\}$ **then**
  return x
return FAIL

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 001$ (added to $A[2]$)
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 00$
$h(x_4) = 00$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then $\text{getAny}(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then $\text{getAny}(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A):

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 001$ (added to $A[2]$)
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 000$
$h(x_4) = 00$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 001$ (added to $A[2]$)
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 000$
$h(x_4) = 000$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 001$ (added to $A[2]$)
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 0001$ (added to $A[3]$)
$h(x_4) = 000$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then getAny$(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then getAny$(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.

  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A)**:**

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 001$ (added to $A[2]$)
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 0001$ (added to $A[3]$)
$h(x_4) = 000?$
$h(x_5) = 1$ (added to $A[0]$)

Getting Any Element

Theorem

For any capacity $n \in \mathbb{N}$ there exists a linear set sketch using $\mathcal{O}(\log n)$ words of space that supports a getAny operation such that (conditioned on a high probability event)

- if $X = \emptyset$ then $\text{getAny}(S(X)) = \text{EMPTY}$
- if $X \neq \emptyset$ then $\text{getAny}(S(X)) \in X \cup \{\text{FAIL}\}$
- if $|X| \leq n$ then $\Pr[\text{getAny}(S(X)) = \text{FAIL}] \leq \frac{1}{2}$.

Construction

- Use array of $A[0..k]$ of $k = \mathcal{O}(\log n)$ sketches for 1-sparse recovery
- use hash function $h : U \to \{0,1\}^k$ and let $\tilde{h}(x) = \text{countLeadingZeroes}(h(x))$.
- insert(x) adds x to $A[\tilde{h}(x)]$.
  Example: $h(x) = \mathbf{000}1011 \to$ added to $A[3]$

Algorithm getAny(A):

```
if ∀i ∈ {0,...,k}: recover(A[i]) = ∅ then return ∅
if ∃i ∈ {0,...,k}: recover(A[i]) is singleton {x} then
    return x
return FAIL
```

Proof (Analysis of the failure event)

Reveal hash bits of elements column-wise until all but one element x^* has a 1.

- for $k \geq 2\log_2(n)$ this happens in a column $i < k$ whp

$h(x_1) = 001$ (added to $A[2]$)
$h(x_2) = 1$ (added to $A[0]$)
$h(x_3) = 0001$ (added to $A[3]$)
$h(x_4) = 000?$
$h(x_5) = 1$ (added to $A[0]$)

- if the ith bit of $h(x^*)$ is zero, then x^* is alone in $A[\tilde{h}(x^*)]$ and can be returned by findElement.
  $\hookrightarrow$ success probability $\geq \frac{1}{2}$

Exercises

Exercise 1

Reduce the failure probability from $\frac{1}{2}$ to $1/n^{42}$.

Exercises

Exercise 1

Reduce the failure probability from $\frac{1}{2}$ to $1/n^{42}$.

Solution

Use $\log(n^{42}) = \mathcal{O}(\log n)$ independent copies of the sketch.

- increases insertion time to $\mathcal{O}(\log n)$
- increases space to $\mathcal{O}(\log^2 n)$

Exercises

Exercise 1

Reduce the failure probability from $\frac{1}{2}$ to $1/n^{42}$.

Solution

Use $\log(n^{42}) = \mathcal{O}(\log n)$ independent copies of the sketch.

- increases insertion time to $\mathcal{O}(\log n)$
- increases space to $\mathcal{O}(\log^2 n)$

Exercise 2

Can we recover X by repeatedly calling getAny(A) and removing the returned element?

Exercises

Exercise 1

Reduce the failure probability from $\frac{1}{2}$ to $1/n^{42}$.

Solution

Use $\log(n^{42}) = \mathcal{O}(\log n)$ independent copies of the sketch.

- increases insertion time to $\mathcal{O}(\log n)$
- increases space to $\mathcal{O}(\log^2 n)$

Exercise 2

Can we recover X by repeatedly calling getAny(A) and removing the returned element?

Solution: NO.

This cannot work. The space of $\mathcal{O}(\log^2 n)$ is information theoretically insufficient to recover a set of $\mathcal{O}(n)$ elements.

Where the argument fails.

- The analysis assumes “fresh” hashing bits.
- After removing getAny(A) the stored set and the hashing bits are correlated.

Sketches of Sets, Vectors and Graphs

1. Set Sketches...

... for Cardinality Estimation

... for Membership Estimation

2. Vector Sketches and the Turnstile Model

... for $\|x\|_2$-Norm Estimation

... for Heavy Hitter Detection

3. Set Sketches for Sparse Recovery

Warm-Up: 1-Sparse Recovery

n-Sparse Recovery

Getting Any Element

Application: Graph Sketch for Spanning Tree Computation

Graph Sketch for Spanning Forest [Ahn, Guha, McGregor, SODA2012]

Theorem

Let $n \in \mathbb{N}$ and $\mathcal{E} = \{\{i, j\} \mid 1 \leq i < j \leq n\}$. A graph is a subset $E \subseteq \mathcal{E}$. There exists a graph sketch supporting a "spanningForest"-operation such that

- the sketch is linear // $S(E_1 \triangle E_2) = S(E_1) \oplus S(E_2)$
- space is $\mathcal{O}(n \log^3 n)$
- spanningForest($S(E)$) succeeds whp

Graph Sketch for Spanning Forest [Ahn, Guha, McGregor, SODA2012]

Theorem

Let $n \in \mathbb{N}$ and $\mathcal{E} = \{\{i,j\} \mid 1 \leq i < j \leq n\}$. A graph is a subset $E \subseteq \mathcal{E}$. There exists a graph sketch supporting a "spanningForest"-operation such that

- the sketch is linear // $S(E_1 \triangle E_2) = S(E_1) \oplus S(E_2)$
- space is $\mathcal{O}(n \log^3 n)$
- spanningForest($S(E)$) succeeds whp

Sketch Construction.

- For $v \in [n]$ and $E \subseteq \mathcal{E}$ define $N(v) := \{e \in E \mid v \in e\}$. // note: set of *edges*
- Independently, for each $1 \leq i \leq \log_2 n$:
  - Choose random linear sketching function $S^{(i)}$ for subsets of $\mathcal{E}$ supporting getAny.
  - for each $v \in [n]$ store $S^{(i)}(N(v))$.

Graph Sketch for Spanning Forest [Ahn, Guha, McGregor, SODA2012]

Theorem

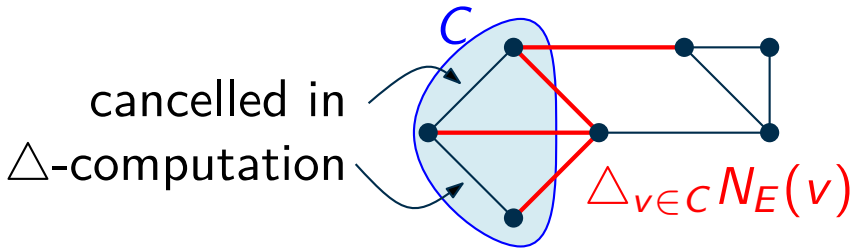
Let $n \in \mathbb{N}$ and $\mathcal{E} = \{\{i,j\} \mid 1 \leq i < j \leq n\}$. A graph is a subset $E \subseteq \mathcal{E}$. There exists a graph sketch supporting a "spanningForest"-operation such that

- the sketch is linear // $S(E_1 \triangle E_2) = S(E_1) \oplus S(E_2)$
- space is $\mathcal{O}(n \log^3 n)$
- spanningForest($S(E)$) succeeds whp

Sketch Construction.

- For $v \in [n]$ and $E \subseteq \mathcal{E}$ define $N(v) := \{e \in E \mid v \in e\}$. // note: set of *edges*
- Independently, for each $1 \leq i \leq \log_2 n$:
  - Choose random linear sketching function $S^{(i)}$ for subsets of $\mathcal{E}$ supporting getAny.
  - for each $v \in [n]$ store $S^{(i)}(N(v))$.

note: for $C \subseteq V$: $\bigoplus_{v \in C} S^{(i)}(N(v)) = S^{(i)}(\underbrace{\triangle_{v \in C} N(v)}_{\text{edges leaving } C})$.



Graph Sketch for Spanning Forest [Ahn, Guha, McGregor, SODA2012]

Theorem

Let $n \in \mathbb{N}$ and $\mathcal{E} = \{\{i,j\} \mid 1 \leq i < j \leq n\}$. A graph is a subset $E \subseteq \mathcal{E}$. There exists a graph sketch supporting a "spanningForest"-operation such that

- the sketch is linear // $S(E_1 \triangle E_2) = S(E_1) \oplus S(E_2)$
- space is $\mathcal{O}(n \log^3 n)$
- spanningForest$(S(E))$ succeeds whp

Algorithm spanningForest$\left(\left(S^{(i)}(N(v))\right)_{v\in[n],1\leq i\leq \log_2 n}\right)$**:**

$F \leftarrow \emptyset$ // known edges

for $i = 1$ **to** $\log n$ **do**

let $\mathcal{C}$ be the connected components of F

// find edge existing each component, if one exists

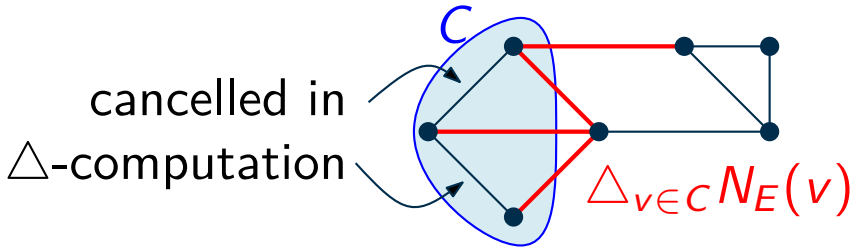
$F \leftarrow F \cup \{\text{getAny}(\bigoplus_{v\in C} S^{(i)}(N(v))) \text{ if not empty} \mid C \in \mathcal{C}\}$

return spanningForest(F)

Sketch Construction.

- For $v \in [n]$ and $E \subseteq \mathcal{E}$ define $N(v) := \{e \in E \mid v \in e\}$. // note: set of *edges*
- Independently, for each $1 \leq i \leq \log_2 n$:
  - Choose random linear sketching function $S^{(i)}$ for subsets of $\mathcal{E}$ supporting getAny.
  - for each $v \in [n]$ store $S^{(i)}(N(v))$.

note: for $C \subseteq V$: $\bigoplus_{v\in C} S^{(i)}(N(v)) = S^{(i)}(\underbrace{\triangle_{v\in C} N(v)}_{\text{edges leaving } C})$.



Graph Sketch for Spanning Forest [Ahn, Guha, McGregor, SODA2012]

Theorem

Let $n \in \mathbb{N}$ and $\mathcal{E} = \{\{i,j\} \mid 1 \leq i < j \leq n\}$. A graph is a subset $E \subseteq \mathcal{E}$. There exists a graph sketch supporting a "spanningForest"-operation such that

- the sketch is linear // $S(E_1 \triangle E_2) = S(E_1) \oplus S(E_2)$
- space is $\mathcal{O}(n \log^3 n)$
- spanningForest$(S(E))$ succeeds whp

Algorithm spanningForest$\left(\left(S^{(i)}(N(v))\right)_{v\in[n],1\leq i\leq \log_2 n}\right)$**:**

$F \leftarrow \emptyset$ // known edges

for $i = 1$ **to** $\log n$ **do**

let $\mathcal{C}$ be the connected components of F

// find edge existing each component, if one exists

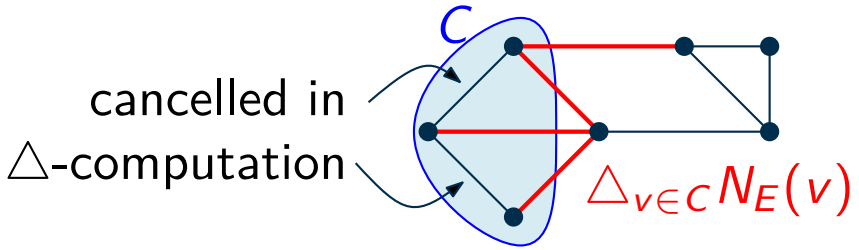
$F \leftarrow F \cup \{\text{getAny}(\bigoplus_{v\in C} S^{(i)}(N(v))) \text{ if not empty} \mid C \in \mathcal{C}\}$

return spanningForest(F)

Sketch Construction.

- For $v \in [n]$ and $E \subseteq \mathcal{E}$ define $N(v) := \{e \in E \mid v \in e\}$. // note: set of *edges*
- Independently, for each $1 \leq i \leq \log_2 n$:
  - Choose random linear sketching function $S^{(i)}$ for subsets of $\mathcal{E}$ supporting getAny.
  - for each $v \in [n]$ store $S^{(i)}(N(v))$.

note: for $C \subseteq V$: $\bigoplus_{v\in C} S^{(i)}(N(v)) = S^{(i)}(\underbrace{\triangle_{v\in C} N(v)}_{\text{edges leaving } C})$.



Exercise: Complete the proof. // discussed in lecture

Remark: There's more!

[Ahn, Guha, McGregor, 2012] achieved the following

Graph sketches using $\mathcal{O}(n \cdot \operatorname{poly}\log n)$ space for

- deciding k-connectivity // we have seen $k = 1$
- approximate weight of *minimum* spanning tree
- deciding bipartiteness

Is this relevant?

- Sketches for spanning forests: Not very.
  - maybe for communication networks
- The general idea of graph sketching: Very much!

Lossy linear projections of graph data can preserve relevant structural information.

Summary

TODO

name	sketch of what?	what for?	$S(X)$	how to decode	notes	linear?
HyperLogLog	set $X \subseteq U$	estimate $\lvert X\rvert$	$\max_{x\in X}$ leadingZeroes$(h(x))$	$\lvert X\rvert \approx 2^{S(X)}$	HLL is harmonic mean of several such sketches	✗
quotient filter	set $X \subseteq U$	estimate $[x \in X]$	$\{f(x) \mid x \in X\}$	$[x \in X] \approx [f(x) \in S(X)]$	false positive rate $\frac{\lvert X\rvert}{\lvert F\rvert}$, no false negatives	✗
AMS	vector $x \in \mathbb{Z}^n$	estimate $\lVert x\rVert_2^2$ or $\langle x, y\rangle$	$\sum_{i\in[n]} h(i)x_i$ where $h : [n] \to \{-1, 1\}$	$\langle x, y\rangle = \mathbb{E}[S(X)S(Y)]$	average several estimates, good for $\lVert x\rVert_2^2$	✓
CountMinSketch	vector $x \in \mathbb{N}_0^n$	estimate x_i	add x_i to t out of t/ϵ counters	$x_i \leq$ min of x_i's counters	detects heavy hitters, x_i overestimated by $\leq \epsilon\lVert x\rVert_1$ with prob $1 - 2^{-t}$	✓
?	set $X \subseteq U$	1-sparse recovery	$(\sum$,hashSum)	$\{sum\}$ if $h(\sum) =$ hashSum	adaptable to multisets	✓
invertible bloom lookup table	set $X \subseteq U$	n-sparse recovery	(not part of this lecture)	(not part of this lecture)	see	✓
graph sketch [AGM]	edge set $E \subseteq \mathcal{E}$	spanning forest recovery	(see slides)	(see slides)	inspired lots of follow up work	✓

Possible Exam Questions

TODO