**Sketching Algorithms for Big Data** Fall 2017

# Lecture 07 — September 21, 2017

*Prof. Jelani Nelson* *Scribe: James Yang*

## 1 Overview

Today: Point query/Heavy hitter

1. $l_1$ point query: $query(i) = x_i \pm \frac{1}{k}\|x_{tail(k)}\|_1$
2. $l_1$ heavy hitters: $heavy()$ outputs $L \in [n]$ such that
   (1) $|x_i| > \frac{1}{k}\|x_{tail(k)}\|_1 \implies i \in L$ ,"i is a $\frac{1}{k}$-heavy hitter"
   (2) $|L| = O(k)$

In this lecture we are going to cover few algorithms on point query and heavy hitters.

## 2 $l_1$ point query

### CountMin sketch [1] (See Figure 1)

1. Hashing $h_1, ..., h_r : [n] \to [t]$, drawn independently from 2-wise family
2. Each grid cell is a counter we store in memory
3. $update(i, \triangle)$, $C_{j,h_j(i)} \leftarrow C_{j,h_j(i)} + \triangle$
4. For now, assume $\forall i, x_i \geq 0$, (strict turnstile assumption)
5. $query(i)$ : output $\tilde{x_i} = \min_{1\leq j\leq r} C_{j,h_j(i)}$

### Analysis

1. Want to show

$$\mathbb{P}(|\tilde{x_i} - x_i| > \frac{1}{k} \cdot \|x_{tail(k)}\|_1) < \eta$$

2. Suffices to show for fixed: $j \in [r]$

$$\mathbb{P}(|C_{j,h_j(i)} - x_i| > \frac{1}{k} \cdot \|x_{tail(k)}\|_1) \leq \frac{1}{2}$$

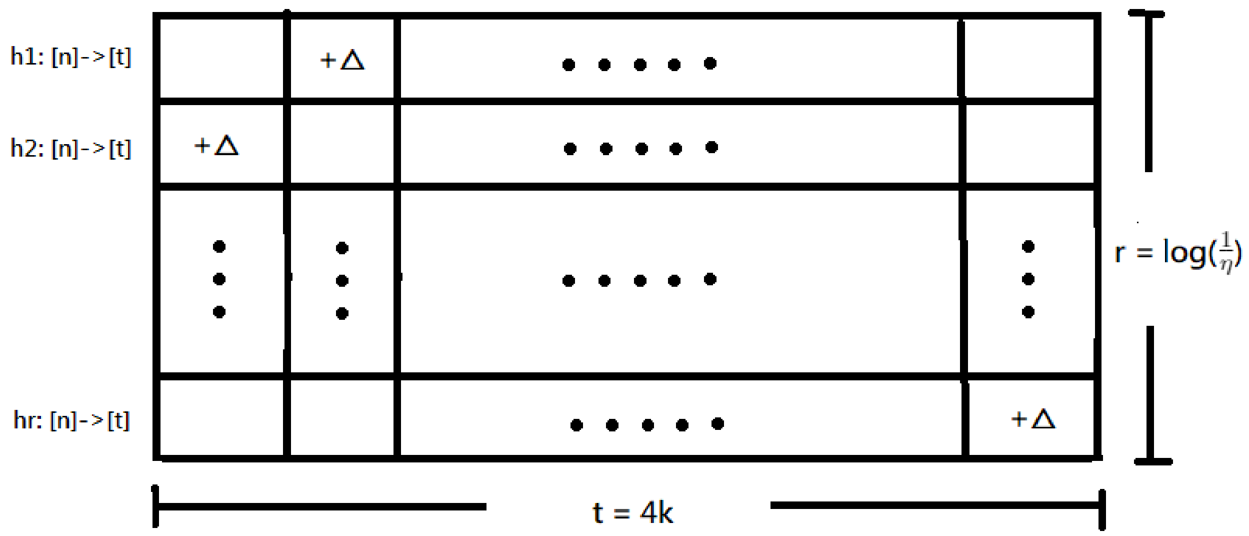


Figure 1: CountMin Sketch

3. Define event:

$$E_i = \text{event that none of the top k entries in } x \text{ collide with } i \text{ under } h_j$$

$$F_i = \text{event that the total tail(k) mass colliding with } i \text{ under } h_j \text{ is } \leq \frac{1}{k}\|x_{tail(k)}\|_1$$

**Claim 1.** *If $E_i \cap F_i$ holds, then $|C_j, h_j(i) - x_i| \leq \frac{1}{k}\|x_{tail(k)}\|_1$*

**Claim 2.** *If $\mathbb{P}(\bar{E}_i \cup \bar{F}_i) \leq \frac{1}{2}$ , where $\mathbb{P}(\bar{E}_i \cup \bar{F}_i) \leq \mathbb{P}(\bar{E}_i) + \mathbb{P}(\bar{F}_i)$*

*Proof.* Note that t=4k

For $\bar{E}_i$ :

Let $\beta_q = \begin{cases} 1 \text{ , if } h_j(q) = h_j(i) \\ 0 \text{ , otherwise} \end{cases}$

$$\mathbb{P}h_j(\bar{E}_i) = \mathbb{P}_{h_j}(\sum_{q \neq i, q \in head(k)} \beta_q \geq 1) \leq \frac{\mathbb{E}(\sum \beta_q)}{1} \leq \frac{1}{4}$$

For $\bar{F_i}$ :

$$\begin{aligned}
\mathbb{P}_{h_j}(\bar{F_i}) &= \mathbb{P}_{h_j}(\text{Total tail mass colliding with i} > \frac{\|x_{tail(k)}\|_1}{k}) \\
&\leq \frac{\mathbb{E}(\text{Total tail mass colliding with i})}{\frac{\|x_{tail(k)}\|_1}{k}} \\
&= \frac{\mathbb{E}(\sum_{q\neq i, q\in tail(k)} x_q \cdot \beta_q)}{\frac{\|x_{tail(k)}\|_1}{k}} \\
&\leq \frac{\frac{\|x_{tail(k)}\|_1}{t}}{\frac{\|x_{tail(k)}\|_1}{k}} \\
&= \frac{k}{t} = \frac{1}{4}
\end{aligned}$$

□

## Point Query

1. Idea: $query(i)$ for each $i$ then return the $i$ with top k $|\tilde{x_i}|$ values.

2. One issue: (Let's define $Err(k) = \frac{1}{k}\|x_{tail(k)}\|_1$)
We have $x_i = Err(k)$ , maybe $\tilde{x_i} = x_i$ (no error). And there is a $x_{i'} = \frac{1}{1.9} \cdot Err(k)$ , and may have error $\tilde{x_i} = x_i + Err(k)$.Therefore, we are going to think that $x_{i'} > x_i$, although it is not true. So, below we are going to show how to fix this issue.

### Fix

1. Run Point Query structure with $k' = 4k$.

2. Return $L =$ top $3k$ indices.

3. Set $\eta = \frac{\delta}{n}$ , n is the length of vector x

**Claim 3.** $\mathbb{P}$(*L satisfies the true top 3k indices*) $\geq 1 - \delta$

*Proof.* $\mathbb{P}(\exists$ i such that $query(i)$ fails$) < \eta \cdot n = \delta$ , where n is union bound

## Remark

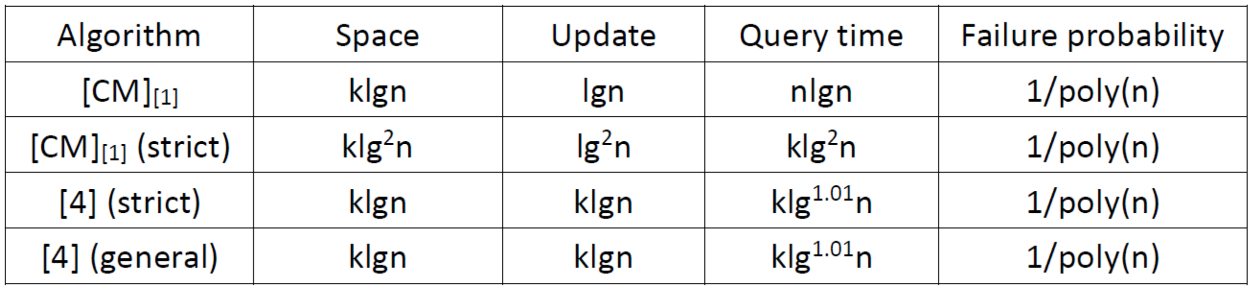
1. A true $\frac{1}{k}$-Heavy Hitter appears as $\geq truevalue - \frac{1}{4k} \geq \frac{3}{4} \cdot \frac{1}{k}$

2. For $i'$ to look bigger, $|x_{i'}|$ hard to be $\geq \frac{1}{2k}$, so $\frac{1}{2k} + \frac{1}{4k} \geq \frac{3}{4} \cdot \frac{1}{k}$

3. # indices $\geq \frac{1}{2k} \cdot \|x_{tail(k)}\|_1$ is $\leq 3k$

$\square$

**Comparison Chart**



| Algorithm | Space | Update | Query time | Failure probability |
|---|---|---|---|---|
| [CM][1] | klgn | lgn | nlgn | 1/poly(n) |
| [CM][1] (strict) | $k\lg^2 n$ | $\lg^2 n$ | $k\lg^2 n$ | 1/poly(n) |
| [4] (strict) | klgn | klgn | $k\lg^{1.01} n$ | 1/poly(n) |
| [4] (general) | klgn | klgn | $k\lg^{1.01} n$ | 1/poly(n) |

Figure 2: Comparison Chart

## 3 $l_2$ - Heavy Hitter / Point Query

Solved by Count Sketch [2]. Structure similar to Figure 1.

1. $\sigma_1, \cdots, \sigma_r$ (2-wise) , $[n] \to \{-1, 1\}$
2. *update*$(i, \triangle)$, $C_{j,h_j(i)} \leftarrow C_{j,h_j(i)} + \triangle$, where $\triangle = \sigma_j(i) \cdot \hat{\triangle}$
3. *query*$(i)$ : output $\tilde{x_i} = \underset{L \leq j \leq r}{\text{median}} \, \sigma_j(i) \cdot C_{j,h_j(i)}$

**Claim 4.** $\mathbb{P}(|\tilde{x_i}(j) - x_i| > \frac{1}{\sqrt{k}} \cdot \|x_{tail(k)}\|_2) < \frac{1}{3}$ *, where* $Err_x(k) = \frac{1}{\sqrt{k}} \cdot \|x_{tail(k)}\|_2$

*Proof.*

$$\mathbb{P}(|\tilde{x_i}(j) - x_i|^2 > Err_2^2(k)) < \frac{\mathbb{E}(\sigma_j(i) \cdot C_{j,h_j(i)} - x_i)^2}{\frac{1}{k}\|x_{tail(k)}\|_2^2} < \frac{1}{3}$$

$\square$

## 4 Insertion only

### $l_1$ $\frac{1}{k}$−Heavy Hitter

For more details, you can look at paper [3]. We use the following example to illustrate the idea of identifying a heavy hitter. Suppose that we have one counter associated with the candidate HH index. If $x_i$ of the stream comes, then the counter is incremented when the i matches with HH.

Otherwise, the counter associated with HH is decremented. If the counter is zero, we reset the candidate HH with the incoming element.

For example, suppose that the input stream is {2, 1, 1}. So 2 comes first and 2 will become the candidate HH which sets the counter to 1. The next element decreases the counter back to 0 (since 1 does not equal to 2). Finally, when the second 1 comes in, the candidate HH is reset to 1 (with counter value 1).

### $l_2-$**Heavy Hitter in insertion only**

Refer to the paper [5], the algorithm BPTree for l2 heavy hitters in insertion-only streams can achieves $O(\epsilon^{-2}\log\epsilon^{-1})$ words of memory and $O(\log\epsilon^{-1})$ update time, which is the optimal dependence on n and m.

The BPTree algorithm runs a series of $\theta$(log n) rounds where the goal of each round is to learn one bit of the identity of Heavy Hitter. Let us use the case of the super heavy hitter for illustrating how BPTree works.

C-super heavy: $x_H^2 > C \cdot \sum_{j\neq i} x_j^2$

We have C0 and C1, are used for keeping the tracks. Learning the super heavy item (H) is done bit-by-bit. When an item comes, the first bit of the item is checked. If it is a 0, then it is send to C0 to do a dot product with random $\pm 1$ vector. Run this for a while, what we are going to expect is that the C corresponding to the super heavy item is bigger, because the item is super heavy. So we shall assume the first bit is 1 if C1 is big. Based on this, bit by bit identification can be done.

## References

[1] Graham Cormode, S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75 , 2005.

[2] Moses Charikar, Kevin C. Chen, Martin Farach-Colton. Finding Frequent Items in Data Streams. ICALP 2002: 693–703.

[3] J. Misra, David Gries. Finding Repeated Elements. *Science of Computer Programming2*, 143–152 , 1982.

[4] K. Larsen, J. Nelson, Huy L.Nguyen, M. Thorup. Heavy hitters via cluster-preserving clustering. *IEEE 57th Annual Symposium on Foundations of Computer Science*, 2016.

[5] V. Braverman, S. Chestnut, N. Ivkin, J. Nelson, Z. Wang, D. Woodruff. BPTree: an l2 Heavy Hitters Algorithm Using ConstantMemory. *36th ACM SIGMOD-SIGACT-SIGAI Symposium*, 2017.