COMS 4995-8: Advanced Algorithms (Spring'21) Jan 28, 2021

Lecture 6: Heavy Hitters and Frequency Moments

Instructor: *Alex Andoni* Scribes: *Ilica Mahajan, Cassandra Marcussen*

1 Overview of Lecture

Today's lecture is a continuation of our discussion about streaming and sketching algorithms. We start by discussing heavy hitters and improvements to the Count Min Algorithm. Then, the lecture progresses to a discussion of computing frequency moments of a stream and the Tug-of-War algorithm for approximating the second frequency moment.

2 Continuing Heavy Hitters

2.1 Overview of Heavy Hitters and the Count Min Algorithm

Last lecture, we discussed heavy hitters. Let f_x be the frequency of x, and $x \in [n]$. We say x is a ϕ-Heavy Hitter (HH) if $f_x \geq \phi$ and $\sum_y f_y = \phi m$. (For example, $n = 2^{32}$ for the case of IP addresses).

Last lecture, we discussed the **Count Min Algorithm**, which is a linear sketch of f_x, the frequency of x. We recall that a linear sketch is a sketch that can be represented as Af, where A is a random matrix that the algorithm chooses. The size of the sketch for the Count Min Algorithm is: $O(\frac{\log n}{\epsilon\phi})$ words.
The Count Min Algorithm returns a set $H \subseteq [n]$ such that with probability $\geq 1 - \frac{1}{n}$:

1. If x is a ϕ-Heavy Hitter, then $x \in H$.
2. If x is not a $(1-\epsilon)\cdot\phi$-Heavy Hitter, then $x \notin H$.

We note that for an x not in these two cases, there is no guarantee of whether x is in H or not.

As a side note, how many bits are there for a "word?" There are $O(\log(m))$ bits. Using the Morris Algorithm, we can do even better, approximately $O(\log(\log(m)))$.

The Count Min Algorithm outputs an estimator, which we call $\hat{f}_x$ for a given x. We calculate the estimator in the following way: $\hat{f}_x = \min_i S_i[h_i(x)]$, where S_i is one of the arrays that we initialize for the Full Count Min Algorithm. We find and output the set H with the algorithm by checking iteratively if for each $x \in [n]$, $\hat{f}_x \geq \phi \cdot m$. If this is satisfied, we add x to H.
What is the issue with this algorithm? The time taken by this is at least n, which is the size of the universe of items. This is a problem, since this n is potentially way too large! For example, if we are considering the space of all IP addresses, this n is of size 2^{34}. Space is at least logarithmic with n but time is most certainly not.

2.2 Improvement to Count Min Algorithm

Given properties of the set H above, what is max $|H|$?

$$|H| \leq \frac{1}{\phi(1-\epsilon)}.$$

The reason this is the maximum size for our set H is that any ϕ-Heavy Hitter takes up a ϕ-fraction of the stream. Different Heavy Hitters will take up different ϕ-fractions of the streams. Similarly, any $(1-\epsilon)\cdot\phi$-Heavy Hitter takes up a $(1-\epsilon)\cdot\phi$-fraction of the stream. So, the number of different heavy hitters must be $\leq \frac{1}{(1-\epsilon)\cdot\phi}$.

Note that $n >> m$, i.e. the size of the universe is much larger than the length of the stream. Since our algorithm takes time n, we have a new goal in mind.

Goal: To compute ϕ-Heavy Hitters in time much less than n.

Theorem 1. *We can get the same guarantees as the Count Min Algorithm (on the set H), using:*

1. $O(\frac{log^2 n}{\phi})$ *time to find ϕ-Heavy Hitters.*
2. $O(\frac{log^2 n}{\epsilon\cdot\phi})$ *space, in words.*

We note that the time to find the ϕ-Heavy Hitters is an improvement from the $O(n)$ runtime of the Count Min Algorithm. However, the new space guarantee is larger.

We will now present the building blocks for the proof of this theorem. This theorem uses Count Min as a black-box and uses one trick to speed up the finding of Heavy Hitters. We think of this like a binary search to find the Heavy Hitters.

Definition 2 (Dyadic Interval)**.** *Assume n is a power of 2. A **dyadic interval** on $[n]$ is an interval $[i\cdot 2^l+1,(i+1)2^l]$, where i is an integer such that $i \in \{0,...,\frac{n}{2^l}-1\}$, and $l = \{0,1,...,log_2 n\}$.*

We will develop the **Fast Count Min** algorithm, which stores $\log_2 n+1$ sketches that are each Count Min. In particular, we let Count Min with index l, denoted CM_l be the Count Min Algorithm whose universe consists of items:

$$[1,2^l],[2^l+1,2\cdot 2^l],...,[i\cdot 2^l+1,(i+1)2^l],....$$

In other words, the universe of CM_l is composed of the dyadic intervals at level l. For CM_l, we define the frequency of a dyadic interval $[i\cdot 2^l+1,(i+1)2^l]$ as:

$$f_{[i\cdot 2^l+1,(i+1)2^l]} = \text{sum of the frequencies of the leaves in the interval.}$$

We can represent the Count Mins at each level using a binary tree. As pictured below, each level of the binary tree corresponds to a Count Min whose universe consists of different dyadic intervals.

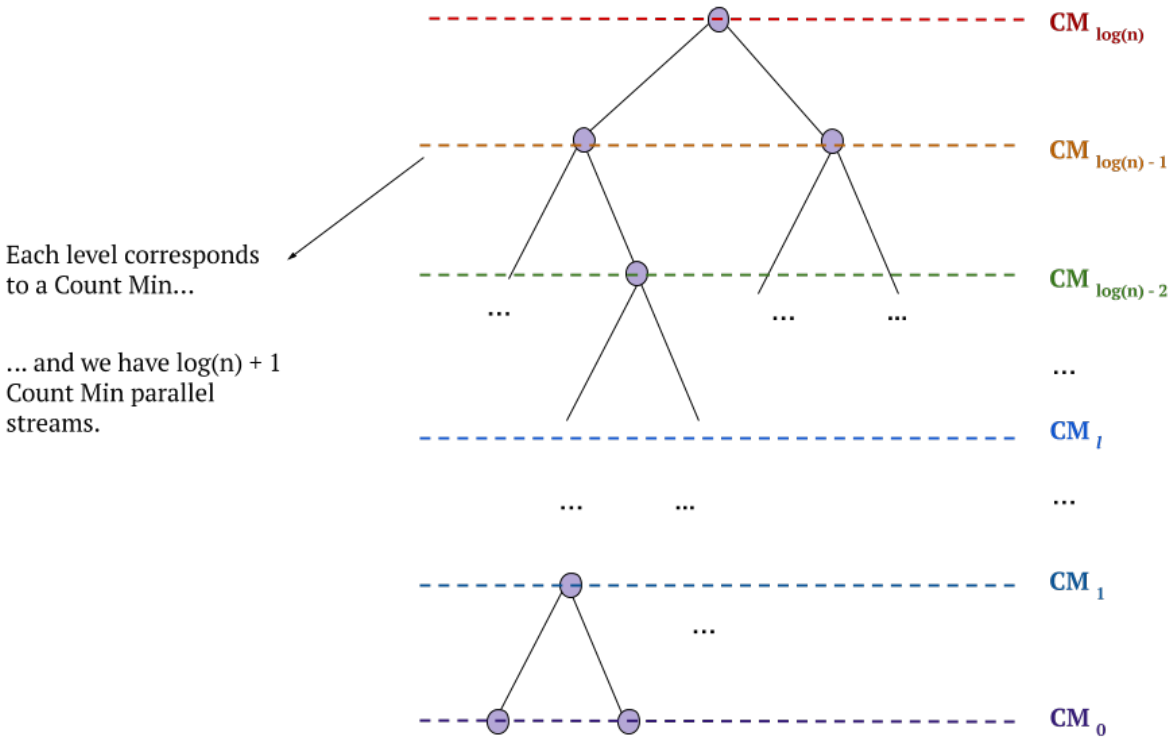

Figure 1: A representation of the Fast Count Min algorithm, where each level l of the binary tree is its own parallel stream, and corresponds to a Count Min.

We can think of our $\log n + 1$ Count Min instances for $[n]$ as having $\log n + 1$ **parallel streams**. Each stream is its own Count Min sketch, independent of the other streams. As we consider Count Min instances over more granular dyadic intervals, we can more closely identify (in terms of number of bits) our heavy hitters. The point is that we are trying to resolve the issue of identifying potential candidates to be a Heavy-Hitter, and can do so using a recursive algorithm where we more closely identify our heavy-hitters using our Count Min instances CM_l for increasing l.

Observation 3. *Fix a diadic interval $I = I_1 \cup I_2$. Then the frequency of I, i.e. $f(I)$ is:*

$$f(I) = f(I_1) + f(I_2).$$

Observation 4. *Fix a diadic interval $I = I_1 \cup I_2$. If I_1 or I_2 is a ϕ-Heavy Hitter, then I is a ϕ-Heavy Hitter.*

Proof. For every level l, $\sum_I f_I = m$, where I is the dyadic interval at level l. So, basically we have the summation of frequencies at the bottom and if I_1 is not a HH then $f_{I_1} \geq \phi m$ then trivially, $f_I = f_{I_1} + f_{I_2} \geq \phi m$

Basically, if a node is a HH then the parent will be a HH too, and the parent is a HH in its own stream. □

Below is a visualization of Observation 4.

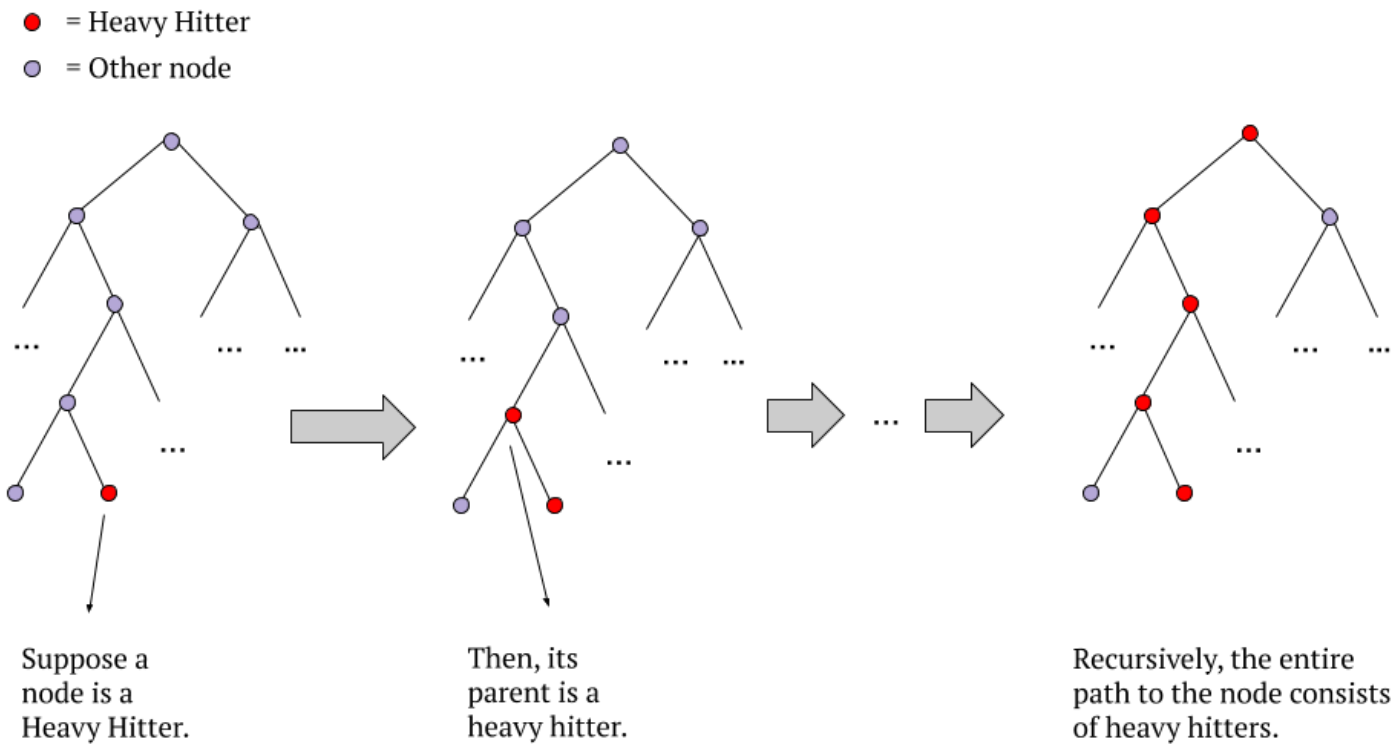

Figure 2: A representation of the property that if a node is a heavy hitter, the its parent will be a heavy hitter. Recursively, this implies that the entire path to a heavy hitter consists of heavy hitters.

Note: Before introducing the improved algorithm to find all of the ϕ-Heavy Hitters, we review three facts we know about our binary tree representing diadic intervals:

1. For a node of level l in the tree, we can check if the node is a ϕ-Heavy Hitter for CM_l.
2. At every level of the binary tree, we have $\leq \frac{1}{(1-\epsilon)\phi}$ Heavy Hitters. This is because every level is a stream, and for each stream a $(1-\epsilon)\cdot\phi$-Heavy Hitter takes up a $(1-\epsilon)\cdot\phi$-fraction of the stream.
3. If a node is a ϕ-Heavy Hitter, then its parent is too.

We introduce the following algorithm to find all the Heavy Hitters:

Algorithm 1: Fast Count Min Algorithm

Result: a data structure using $O(\frac{\log^2 n}{\epsilon \cdot \phi})$ words of space, $O(\frac{\log^2 n}{\phi^2})$ estimation time for $\hat{f}_I$.
HH(node v):

1. If v is leaf and a Heavy Hitter, add v to H, **return**
2. Let c_1 and c_2 be the children of v.
3. If c_1 is a Heavy Hitter, **return HH**(c_1).
4. If c_2 is a Heavy Hitter, **return HH**(c_2).

Remember that the root is always a HH and the interval of the root is $[n]$. This is because $f_{[n]} = m \geq \phi \cdot m$. Also, we check if v, c_1, and c_2 are Heavy Hitters by using the Count Min of their corresponding levels.

2.2.1 Analysis of the Fast Count Min Algorithm

Space: $O(\log(n))$ Count Min sketches which gives $O(\frac{\log^2 n}{\epsilon\phi})$.
Estimation Time: On each level, we have $\leq \frac{1}{(1-\epsilon)\cdot\phi}$ Heavy Hitters. Our estimation time is found as follows:

calls to HH est. of $\hat{f}_I$ = (num. levels of binary tree)·(2 children of each node)·(max num. of HHs per level)

$$\leq O(\log n) \cdot 2 \cdot \frac{1}{(1-\epsilon)\cdot\phi}$$

Hence, overall the time is $O(\frac{\log^2 n}{\phi})$.

3 Frequency Moments and the Tug-of-War Algorithm

3.1 Frequency Moments

We now move to a discussion of the frequency moments of a stream. Before, in our discussion of Heavy Hitters, we were primarily motivated by wanting to find items of *maximum frequency*. In doing so, we wanted to compute $||f_\infty|| = \max_x f_x$.

We now broaden our discussion to consider other norms of f. We can define of p-norm $||f_p||$ as:

Definition 5 (p-norm)**.** *The p-norm of the frequency vectors f_x of the stream is defined as:*

$$||f_p|| := (\sum_x f_x^p)^{1/p}.$$

In particular, we see that $||f_1|| = \sum_x f_x = m$, where m is the length of the stream. We see that $||f_2|| = (\sum_x f_x^p)^{1/p}$. This quantity, $||f_2||$, will be the main focus of our discussion in this section.

Based on the p-norm, we also define frequency moments.

Definition 6 (p-th Frequency Moment)**.** *The p-th frequency moment is defined as:*

$$F_p := ||f_p||^p = \sum_x f_x^p.$$

In particular, we see that $F_2 := ||f_2||^2 = \sum_x f_x^2$.

As a side note, the most natural norms to consider will be the 2-norm and the ∞-norm (correspondingly, we look at the second and infinite frequency moments). The 2-norm is the most important and mathematically the "nicest". This is because it corresponds to Euclidean geometry, or the geometry of the real world. It is also the only norm that has an inner product. While the 1-norm corresponds to *singletons*, the 2-norms corresponds to pairs and relationships between items. We also will see that the 2-norm is interesting and special in a lot of applications. For example, it has applications in least square regression. Also, in the context of random variables, we see that the standard deviation is the 2-norm of deviation from the average.

3.1.1 Motivating the Tug-of-War Algorithm

We now ask: how can we compute F_2 or $||f_2||$? We consider the following example. Suppose we have two frequency vectors, $f_1 = (1, 1, ..., 1)$ and $f_2 = (1, 1, ..., 1, \sqrt{n}, 1, ...1)$. The F_2 norm of f_1 is n. The F_2 norm of f_2 is $2n - 1$. Unfortunately, we cannot distinguish f_1 from f_2 using Count Min unless space is $O(\sqrt{n})$. *Why?* This is because the one term $\sqrt{n}$ in f_2 is a $\frac{1}{\sqrt{n}}$-Heavy Hitter. Our Count Min algorithm may map a lot of values with frequency 1 in the same bucket as this element, particularly if we map to a small table. In the extreme case, for example, if we have only one bucket, the frequencies of 1 overwhelm the $\sqrt{n}$ frequency, and it will be tough to distinguish our two frequency vectors. We'll need a table of at least space $\sqrt{n}$ then.

This motivates the following idea: we can multiply all of our frequencies in a frequency vector by a *random sign* (i.e. by $+1$ or -1). We do this because the sum of n random ± 1 values has expectation 0, and a deviation of $\approx \sqrt{n}$. So, our sum of n random ± 1 values will not be too far from 0. If we have any outlying frequencies, or nonrandom frequencies, then these will influence our new sum. For example, for $f_2 = (1, 1, ..., 1, \sqrt{n}, 1, ...1)$ from our example above, our new sum multiplied with random signs will be proportional to $\approx \pm\sqrt{n}$. This is the motivation behind the main "trick" that the Tug-of-War Algorithm relies on.

3.2 Tug-Of-War Algorithm

Algorithm 2: Tug-of-War Algorithm

Result: an estimator for the second moment z^2.

1. Initialize $z = 0$.
2. Get a hash function, $\sigma : [n] \to \{\pm 1\}$, note that it should be random whether or not we get a +1 or -1.
3. Sketch: $z = \sum_x \sigma_x f_x$, and at seeing an item x, z is updated: $z := z + \sigma_x$
4. Output the estimator for the second moment: z^2.

Analysis: Is the estimator even a good one?

Claim 7. $\mathbb{E}[z^2] = F_2$.

Proof.

$$\mathbb{E}[z^2] = \mathbb{E}[(\sum_x (\sigma_x f_x))^2]$$

$$= \mathbb{E}[\sum_{x,y} \sigma_x \sigma_y f_x f_y]$$

$$= \sum_{x,y} \mathbb{E}[\sigma_x \sigma_y] f_x f_y.$$

Notice here that if $x = y$, $\mathbb{E}[\sigma_x \sigma_y]$ will be 1, and if they are not equal, $\mathbb{E}[\sigma_x \sigma_y]$ will be 0. So,

$$\sum_{x,y} \mathbb{E}[\sigma_x \sigma_y] f_x f_y = \sum_x f_x f_x = F_2.$$

□

Claim 8. $Var[z^2] \leq O(F_2^2)$.

Proof. $Var[z^2] = \mathbb{E}[z^4] - (\mathbb{E}[z^2])^2$.
We can drop the second term here since it is proportional to the first term. So, continuing,

$$\rightarrow = \mathbb{E}[(\sum_x \sigma_x f_x)^4]$$

$$= \sum f_x f_y f_p f_q \cdot \mathbb{E}[\sigma_x \sigma_y \sigma_p \sigma_q]$$

Again, let's think about what happens when $x = y = p = q$ or two pairs, versus when none of them are equal or there is only one pair. In the latter case, the expected value would equal 0. But, if there are two pairs, or $x = y = p = q$, then it will not be 0. There are 3 ways for that to occur.

$$Var[z^2] = \sum_x f_x^4 + 3 \cdot \sum_{x,y} f_x^2 f_y^2$$

$$= (\sum f_x)^2 + 3 \cdot (\sum_x (f_x)^2)^2.$$

□

Thus, we have shown that:

$$\mathbb{E}[z^2] = F_2$$

$$Var[z^2] \leq 4 \cdot F_2^2$$

And using Chebyshev,

$$Pr\left[|z^2 - F_2| > 3 \cdot \sqrt{4F_2^2}\right] \leq \frac{1}{9}$$

and $z^2 = F_2 \pm 6F_2$ with probability $\geq \frac{8}{9}$.

3.3 Tug-Of-War+ Algorithm

To get a better, more concentrated approximation (i.e. for a $(1+\epsilon)$-factor approximation, we use a technique we have seen before and just *repeat* the Tug-of-War algorithm $k = \theta(\frac{1}{\epsilon^2})$ times. This is exactly what the Tug-of-War+ algorithm, introduced below, will do.

Algorithm 3: Tug-of-War+ Algorithm

Result: an estimator z^2.

1. Keep $k = \theta(\frac{1}{\epsilon^2})$ counters $z_1, ...z_k$.

   Each $z_i := \sum_x \sigma_x^i f_x$, where $\sigma^i : [n] \to \{\pm 1\}$ is i.i.d. random.

2. Output the estimator: $z^2 = \frac{1}{k}\sum_{i=1}^{k} z_i^2$.

Claim 9. $\mathbb{E}[z^2] = F_2$.

Proof. This will overall be the same proof as for the Tug-of-War algorithm, also using properties such linearity of expectation as we did in Lecture 1. □

Claim 10. $Var[z^2] \leq \frac{1}{k} \cdot 4 \cdot (F_2)^2$.

Proof. Same proof structure as the proof of the Variance of the random variable X for the Morris Algorithm from Lecture 1. □

Claim 11. *The Tug-of-War+ algorithm provides a $(1 \pm \epsilon)$-approximation of F_2 with probability $\geq 8/9$.*

Proof. We recall Chebyshev's inequality, which tells us that for a random variable X:

$$\Pr[|X - \mathbb{E}[X]| > \lambda] \leq \frac{Var[X]}{\lambda^2}.$$

We set $\lambda = 3 \cdot \sqrt{Var[X]} \leq 3 \cdot \sqrt{\frac{1}{k} \cdot 4 \cdot (F_2)^2}$. Thus, by Chebyshev's inequality, we get:

$$\Pr\left[|X - \mathbb{E}[X]| > 3 \cdot \sqrt{\frac{1}{k} \cdot 4 \cdot (F_2)^2}\right] \leq \frac{\frac{1}{k} \cdot 4 \cdot (F_2)^2}{9 \cdot \frac{1}{k} \cdot 4 \cdot (F_2)^2} \leq \frac{1}{9}.$$

Thus, our "bad event" that $z^2 = F_2 \pm \frac{6}{\sqrt{k}} F_2 = F_2 \cdot (1 \pm \frac{6}{\sqrt{k}})$ occurs with probability $\leq \frac{1}{9}$. We can therefore set $k = \frac{36}{\epsilon^2}$ to get our desired result.

We have therefore found that the Tug-of-War+ algorithm estimates the second frequency moment with up to a $(1 \pm \epsilon)$-factor with probability $\geq 8/9$. That is, our estimator $z^2 = (1 \pm \epsilon) \cdot F_2$ with probability $\geq 8/9$. □