

**Probability for Computer Science**  
**Prof. Nitin Saxena**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology - Kanpur**

**Module - 8**  
**Lecture - 32**  
**Streaming Algorithms II**

So, in the algorithm, this output is 2 raised to  $T + 0.5$ ; maximum value of  $z$  is  $T$ . So, that is our  $\hat{d}$ .

(Refer Slide Time: 00:21)

$$\triangleright \text{var}(Y_2) = \sum_{j \in \sigma} \text{var}(X_{2,j}) \leq \sum_{j \in \sigma} E[X_{2,j}^2]$$

$$\stackrel{\text{(i.i.d. hash)}}{=} \sum_{j \in \sigma} E[X_{2,j}] = E[Y_2] = d/2^z.$$

- Output is  $\hat{d} := 2^{T+0.5}$ .
- Let  $a \in \mathbb{Z}$  be the smallest:  $2^{a+0.5} \geq 3d$ .
- Let  $b \in \mathbb{Z}$  " " largest:  $2^{b+0.5} \leq d/3$ .

$$\triangleright P(\hat{d} \geq 3d) = P(T \geq a) = P(Y_a > 0) = P(Y_a \geq 1)$$

$$\leq E[Y_a]/1 = d/2^a \leq \frac{\sqrt{2}}{3} < 1.$$

Now, let us compare it with  $3d$  and  $d$  by 3. So, let  $a$  be the smallest integer, such that 2 raised to  $a + 0.5$  is just above  $3d$ . It is kind of log; we are looking at log of  $3d$ . And similarly, what symmetric thing is; let  $b$  be the largest, such that 2 raised to  $b + 0.5$  is just behind  $d$  by 3. This is again,  $d$  is kind of log of  $d$  by 3. Now, we want to compare  $T$  with  $a$  and  $b$ ; how much of difference is possible?

So, first is the probability that  $\hat{d}$  exceeds  $3d$ , which means that  $T$  exceeds  $a$ . So, what is that? So, if  $T$  exceeds  $a$ , then basically  $Y_a$  is positive. And positive means,  $Y_a$  can only be an integer; so, it is at least 1. Now, what is the probability that this random variable  $Y_a$  is 1 or large. So, there we use concentration inequality. So, you can write this as bounded by expectation of  $Y_a$  by 1. That is Markov inequality.

So, expectation, you understand,  $d$  over  $2$  raised to  $a$ . Now, what is  $d$  over  $2$  raised to  $a$ ? So, the way we defined  $a$ , we actually get a bound, square root  $2$  by  $3$ ; that is less than  $1$ . So, the chance that  $\hat{d}$ , the output exceeds  $3d$ , it is more than  $3$  times  $d$  is sufficiently away from  $1$ . This is what we learn. And similarly, we can do the other part.

**(Refer Slide Time: 03:32)**

$$\begin{aligned} \Delta P(\hat{d} \leq d/3) &= P(T \leq b) = P(Y_{b+1} = 0) \\ &\leq P(|Y_{b+1} - E[Y_{b+1}]| \geq d/2^{b+1}) \leq \frac{\text{var}(Y_{b+1})}{(d/2^{b+1})^2} \\ &= 2^{b+1}/d \leq \sqrt{2}/3. \end{aligned}$$

$$\Delta P\left(\frac{d}{3} \leq \hat{d} \leq 3d\right) \geq 1 - \frac{2\sqrt{2}}{3} > 0.$$

**Boosting:** The prob. can be made  $1 - 2^{-\Omega(k)}$  by running  $k$  independent (A parallel) copies of the algo. for the stream  $\sigma$ . And OUTPUT the Median.

**Chernoff. bd. gives**  
**Thm 1:**  $P(d/3 \leq \text{output} \leq 3d) \geq 1 - 2^{-\Omega(k)}$  &  $\Omega \leq k \cdot \lg n$ .

So, probability that  $\hat{d}$  is less than equal to  $d$  by  $3$ , this is probability that;  $\hat{d}$  less than equal to  $d$  by  $3$  means that  $T$  is less than equal to  $b$ , which means, the way we define  $Y$ , it means that  $Y_{b+1}$  is  $0$ . And what concentration inequalities should be used here? So, we can use here Chebyshev inequality. So, you need knowledge of variance and expectation. So, probability that  $Y_{b+1}$  minus the expectation of  $b+1$ ;  $Y_{b+1}$  is  $0$  and expectation is  $d$  over  $2$  raised to  $b+1$ ; so, there is enough gap.

That is what we will utilise. So, what is the chance that there is this gap between  $Y_{b+1}$  and its expectation? So, that chance is limited by variance, which is; so, variance is in fact this  $d$  over  $2$  raised to  $b+1$ . So, this is  $2$  raised to  $b+1$  by  $d$ . And  $2$  raised to  $b+1$  by  $d$  comes out to be again square root by  $3$ . So, in other words, what you have now is probability that  $\hat{d}$  is smaller than a saviour; it is between  $3d$  and  $d$  by  $3$ .

This is at least  $1 - 2 \sqrt{2}$  by  $3$ . That is positive. So, there is a good chance that the algorithm is actually outputting and a good estimate for  $d$ ; but this probability is not close to  $1$ , right? It is more like close to  $0$ , although it is a constant, positive constant. You can actually boost it by the boosting trick. So, the probability can be made  $1 - 2$  raised to minus  $k$ , which means as close to  $1$  as you want.

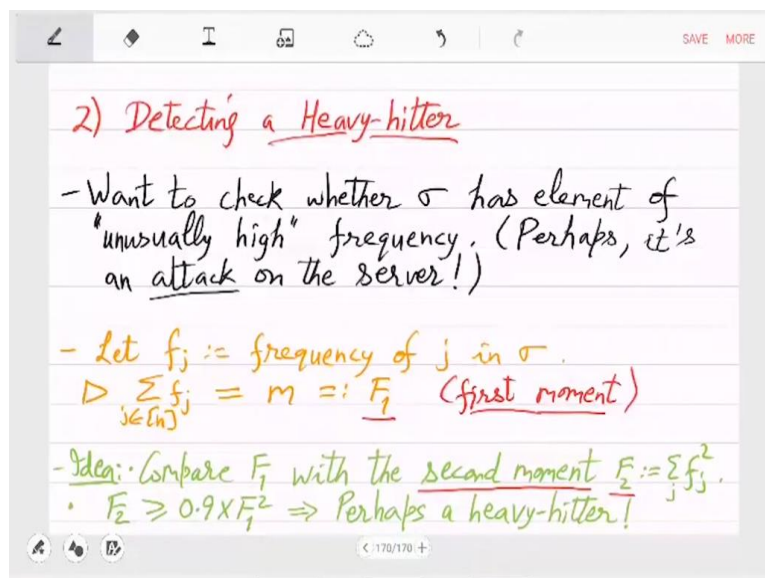
You can boost the success probability by running  $k$  independent and parallel copies of the algorithm for the stream. The independence part is important, which means that instead of picking 1 hash function, you pick  $k$  hash functions completely independently, and then use this valuation trick. So, you will have  $k$  many  $T$ 's that you will get in the end. And what do you output as  $\hat{d}$ ?

So, there is a nice trick; you output the median of these  $k$  answers. So, not the least  $T$  or the maximum  $T$  out of these  $k$ , but the median. So, the advantage is that, you can show that the median being far away from  $d$ , that probability you can bound by Chernoff bound. So, Chernoff will give you theorem that the output is between  $d/3$  and  $3d$  is quite high, as high as you want. And how much space is needed?

Well,  $\log n$  was for the algorithm; so, it is  $k$  times  $\log n$  now. So,  $k$ , you can take to be 10. And then, the success probability is almost 1 for the output approximating the number of distinct elements, while the space is still constant times  $\log n$ . So, that is an amazing algorithm. When we started, it was not clear whether such a thing will exist, but it does. So, that is the first algorithm in data streaming.

Second question that we will take up is whether there is an element in the stream that has a very high frequency, which basically suggests; in applications, it could suggest an attack, some token is appearing too many times; too many requests coming from that client.

(Refer Slide Time: 10:05)



2) Detecting a Heavy-hitter

- Want to check whether  $\sigma$  has element of "unusually high" frequency. (Perhaps, it's an attack on the server!)
- Let  $f_j :=$  frequency of  $j$  in  $\sigma$ .  
 $\triangleright \sum_{j \in [n]} f_j = m =: F_1$  (first moment)
- Idea: Compare  $F_1$  with the second moment  $F_2 := \sum_j f_j^2$ .  
•  $F_2 \geq 0.9 \times F_1^2 \Rightarrow$  Perhaps a heavy-hitter!

So, detecting a heavy hitter. So, we want to check whether  $\sigma$  has elements of unusually high frequency? So, perhaps, it is an attack on the server. So, when you have a server and a client and many clients are making a request, this algorithm can be used, can be seen as a way to detect whether 1 client is sending too much of data, too many requests compared to other clients. So, then you can decide to just ban that client.

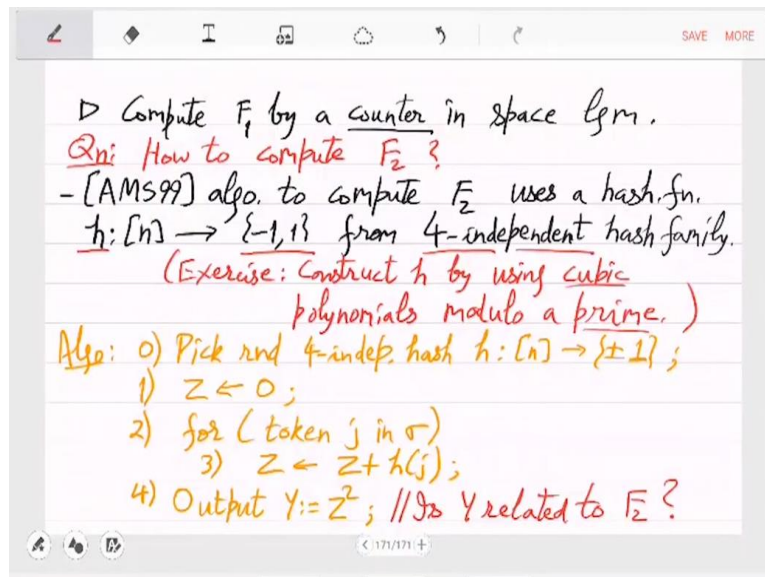
So, this client is called heavy hitter. How do you detect a heavy hitter? Again, in a very long data stream; you cannot really remember what that client did, because there are also many clients and there are many requests. So, in terms of data stream model, what you want is, now you want to work with frequencies. So, let  $f_j$  be the frequency of  $j$  in the stream  $\sigma$ . So, what is summation of all the frequencies? So, summation of frequencies is  $m$ .

That is the length of the stream. So, we call it the first moment,  $F_1$ . And the idea is, for the algorithm, compare  $F_1$  with the second moment, which is sum of the square of the frequencies. So, heuristically, if  $F_2$  is very close to  $F_1^2$ , let us say 0.9 times  $F_1^2$ , then it suggests that there is essentially 1 frequency which is very large compared to the others.

Obviously, if there is only 1 kind of token, then  $F_2$  is  $F_1^2$ ; but if there are many tokens, then  $F_2$  cannot really be equal to  $F_1^2$ ; it will always be smaller; but it being close to  $F_1^2$ , suggests that something is wrong; there is a heavy hitter; implies that perhaps a heavy hitter. So, comparison is fine. You can decide how to define heavy hitter in terms of  $F_2, F_1$ .

This is just 1 option. But the bigger question is, how do you compute  $F_1$  and  $F_2$ ? So,  $F_1$ , you can compute just by keeping a counter. It requires  $\log m$  space. How do you compute  $F_2$ ? That is the tough question.

**(Refer Slide Time: 14:54)**



So, compute  $F_1$  by a counter in space  $\log m$ . This is fine. The major question here, the only question in this problem is, how do you compute  $F_2$ ? So,  $F_2$ , the problem is that, this is a non-linear expression. So, you cannot really divide it into token by token computation. This seems to need  $F_j$  in full. So, unless you have reached the end of the stream and you have calculated all the  $F_j$ 's,  $F_1$  to  $F_n$ , it is not really clear how do you compute the nonlinear expressions.

So, counter is not clear. This indeed seems impossible to compute exactly. So, we will see an amazing algorithm again, which is by Alon, Matias and Szegedy. So, the AMS99 algorithm to compute  $F_2$  uses a hash function  $h$ , like the previous algorithm; but unlike the previous algorithm, this will have a range plus minus 1, which is very strange. It actually will assign a sign to a token, plus sign or minus sign.

And hash function, the sampling condition is also tougher, it needs 4-independence; from 4-independent hash family. So, pairwise independent hash family construction, we have seen in this course. How will you actually construct 4-wise independent hash family? You really have to go beyond matrices for that. So, I will just leave as an exercise; by using cubic polynomials modulo a prime. So, it will be a different construction.

These things exist; you can study them independently. AMS algorithm says that you pick a random 4-independent hash function  $h$ , which assigns signs to your  $n$  tokens appearing in this long stream of  $m$  tokens.  $h$  will be very compactly stored; you do not need  $h$  values on 1 to  $n$ .

It will just be this cubic polynomial. So, it will be very compact. And you can do computations as you see tokens in your stream. That is the beauty of it.

So, first pick random 4-independent hash  $h$  compactly stored. Initialise  $Z$  to 0, for token  $j$  in the stream. This is similar to the algorithm we did before. So, when you see token  $j$  in the stream, you compute the hash function and keep adding. That is it. So, you either will add 1 or minus 1 depending on image of  $h$ . So, you are actually adding these plus minus 1s; so, in the end, you may also have a negative number.

Now, what is the meaning of this  $Z$ ? It is completely unclear; some magical thing. So, the output, you simply output the square of this. So,  $Z$  may be negative, may be positive, but when you square it, it is always non-negative, and that is what you output. Now, the question is, what is this to do with  $F_2$ ? So, is  $Y$  related to  $F_2$ ? That is not at all clear in this algorithm. It is a very simple algorithm. But now, all the work will be done in the analysis.

**(Refer Slide Time: 21:09)**

The image shows a digital whiteboard with the following handwritten text:

Analyse:  $\triangleright Z = \sum_{j \in [n]} f_j \cdot h(j)$ .

$\triangleright E[Z] = \sum_j f_j \cdot E[h(j)] = 0$ .

$\triangleright E[Y] = E\left[\left(\sum_j f_j \cdot h(j)\right)^2\right] = E\left[\sum_j f_j^2 + \sum_{i \neq j} f_i \cdot f_j \cdot h(i) \cdot h(j)\right]$

$= F_2 + \sum_{i \neq j} f_i \cdot f_j \cdot E[h(i)] \cdot E[h(j)] = F_2$ .

*\* 2-wise indep. h*

$\Rightarrow$  We're on the right track!

So, first property of  $Z$  is; which is actually the definition already in the algorithm, that for the stream; or let me say  $j$  1 to  $n$ . So, for these tokens,  $h_j$ , image of  $j$  under the hash function, this will be added as many times as the element appears. So, that is  $f_j$ ; that is by definition. That is what the algorithm is doing. And so, what is the expectation of this? By linearity of expectation, you get; frequency is fixed, that is not a random variable;  $h_j$  is; but since  $h$  is randomly picked,  $h_j$  is either either 1 or minus 1 with equal probability. So, this is 0.

So, expectation of Z is, what you expect Z to come out is 0. So, can you say the same thing about Y? Is Y also expected to be 0? Well, not quite. So, that is what you will now see. It is a different game altogether. So, expectation of Y is expectation of this thing squared, Z squared. So,  $\sum_{i \neq j} f_i f_j$  squared. So, that is expectation of, that inside thing is  $\sum_{i \neq j} f_i f_j$  square plus  $\sum_{i \neq j} f_i f_j$ ; i different from j in  $h_i h_j$ .

So, expectation of  $\sum_{i \neq j} f_i f_j$  square is just  $F^2$ , right? So, that is  $F^2$  plus; interesting thing is the second thing. So, here, by linearity of expectation, you take it inside. And in fact, I also factorise; I express it as  $h_i$  times  $h_j$ ; expectation of first times expectation of the second. Why did I do that? So, this I can do by; since i and j are different and hash function is 4-wise independent; in fact, so, it is 2-wise independent.

So, because of that, I can factorise E; and then, again, this part is 0. So, you see that this justifies our algorithm; because all those Z is expected to be 0, Z square is not. And this should be a big surprise, right? Somehow this probabilistic calculation is telling you that Z square is expected to be what you wanted, which is  $F^2$ . But now, what is the guarantee that it will be close to  $F^2$  in practice? For that, we need concentration inequalities. So, let us do that. But this at least tells you that we are on the right track.

**(Refer Slide Time: 25:40)**

The image shows a digital whiteboard with the following handwritten mathematical derivation:

$$\begin{aligned} \triangleright \text{Var}(Y) &= E[Y^2] - E[Y]^2 = E[Z^4] - E[Y]^2 \\ &= E\left[\left(\sum_{i \neq j} f_i h_j\right)^4\right] - F^2 \\ &= E\left[\sum_j f_j^4\right] + 3 \cdot E\left[\sum_{i \neq j} f_i^2 f_j^2\right] - F^2 \\ &\quad \text{R 4-wise-indep. of } h \\ &= \sum_j f_j^4 + 3 \cdot \sum_{i \neq j} f_i^2 f_j^2 - \sum_j (f_j^2)^2 - \sum_{i \neq j} f_i^2 f_j^2 \\ &= 2 \cdot \sum_{i \neq j} f_i^2 f_j^2 \leq 2 \cdot F^2 \\ \triangleright \text{Chebyshev} &\Rightarrow P(|Y - E| > \alpha \cdot F) \leq \frac{2F^2}{(\alpha F)^2} = 2/\alpha^2 \\ &\Rightarrow Y \text{ approx } F \text{ well!} \end{aligned}$$

So, next what we will do is, for concentration inequality, we will compute the variance of Y. So, variance is Y square minus expectation of Y whole square, which is expectation of Z to the 4 minus this. So, expectation of Z to the 4 is, it is a complicated thing. So, this is  $\sum_{i \neq j} f_i f_j$  to the 4 minus; expectation of Y, we have calculated, that is  $F^2$ . Now, the thing is that when

you expand this out,  $f_j h_j$  to the 4, what you get is  $F_1$  times  $F_2$  times  $F_3$  times  $F_4$ ; 4 things.

So, if they are different, then, point is that expectation will factorise. If they are not different, then it will not factorise. So, when it factorises, you get 0; when it does not factorise, then you are stuck with it. So, you will be stuck with the following terms. So, expectation of  $f_j$  to the 4, you cannot do anything; it is non-zero. And second is, out of 4 things, repetition is there. So, let us say  $F_1$  square times  $F_2$  square; those kinds of things.

So, that is  $f_i$  square  $f_j$  square. And there is because of this binomial expansion, you will have a 3 setting outside, minus  $F_2$  square part. How did we get this? So, this we got by using 4-wise independence. This is really using 4-wise independence. That is where we need 4-wise independence, because you can have in the binomial expansion,  $F_1$  times  $F_2$  times  $F_3$  times  $F_4$ . So, you need expectation of this to be 0.

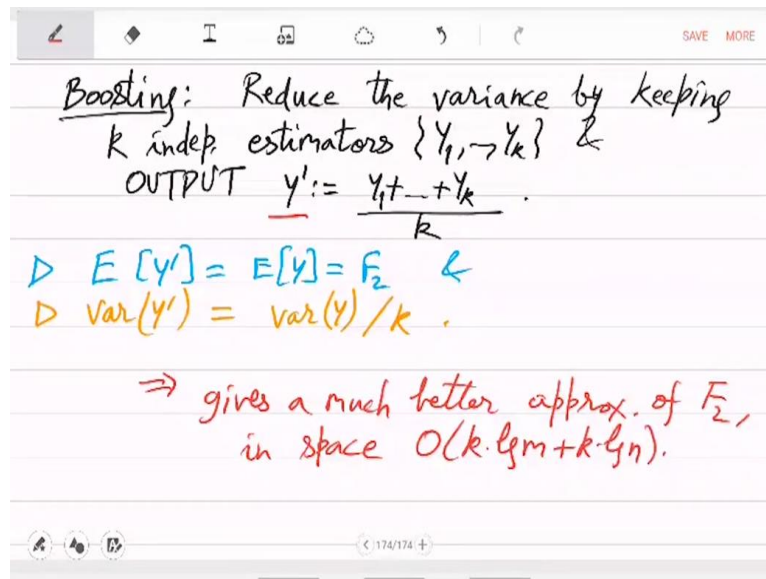
So, you actually need them; expectation to factorise over this, so that it is 0. Now, we have this highly simplified expression, and let us work it out. So, this is equal to; we can take out expectation because these things are actually fixed. So, you have  $\sum f_j$  to the 4, then  $\sum f_i$  square  $f_j$  square minus the green term, which is what? So,  $\sum f_j$  square and  $f_i$  square  $f_j$  square. That is what  $F_2$  square is.

So, what you are left with is just 2 times  $f_i$  square  $f_j$  square. So, that is variance. And this you can upper bound by  $2 F_2$  square. This is, you can see,  $f_i$  square times  $f_j$  square is kind of coming from  $F_2$  times  $F_2$ . So, we have this clear upper bound. And with that, when you use Chebyshev; so, Chebyshev concentration inequality, it gives you that probability that the output  $Y$  is away from the value that we want,  $F_2$ . How much away?

Some  $\alpha$  times  $F_2$  away. So, that is variance of  $Y$ , which is  $2 F_2$  square divided by  $\alpha F_2$  square. So,  $Y$  minus  $F_2$ ,  $Y$  being away from  $F_2$  by let us say  $2$  times  $F_2$ , that probability is no more than half. So, this is again going in a very good direction. It is telling you that you will have an approximation for  $F_2$ . So, this means that  $Y$  approximates  $F_2$  well. So, now we have guarantees. And the guarantee, you can improve further as we did before. You can improve it further by running this algorithm independently and simultaneously.

**(Refer Slide Time: 32:01)**





So, again, what you can do is reduce the variance by keeping  $k$  independent estimators  $Y_1$  to  $Y_k$ . Instead of computing just a single  $Y$ , you compute  $k$  independent ones, obviously by picking  $k$  independent hash functions, 4-wise independent hash functions and output. So, in the previous algorithm, in example 1, you outputted median. Here, what you want is this Chebyshev bound to improve. So, you want a smaller variance.

So, what you do is, you actually take an average of  $Y_1$  to  $Y_k$ . So, output  $Y'$  which is the average; so, not the median; output the average. So, this has the desired effect of expectation being unchanged and variance being; so, variance, remember requires squaring. So, actually, you can see, you can show that this variance has reduced by  $k$ . So, expectation does not change; variance becomes smaller; and hence, when you use Chebyshev inequality, the RHS will be now much smaller.

So, you will get the desired probability. And you can take  $\alpha$  to be as small as you want. So,  $Y'$  will be very good approximation of  $F_2$ . So, gives a much better approximation of  $F_2$  in space  $k$  times  $\log m$ . So, that is it. This finishes streaming algorithms; and in fact, this finishes the course also. So, I hope that you like the theory and you also appreciated the applications. Okay, thanks.