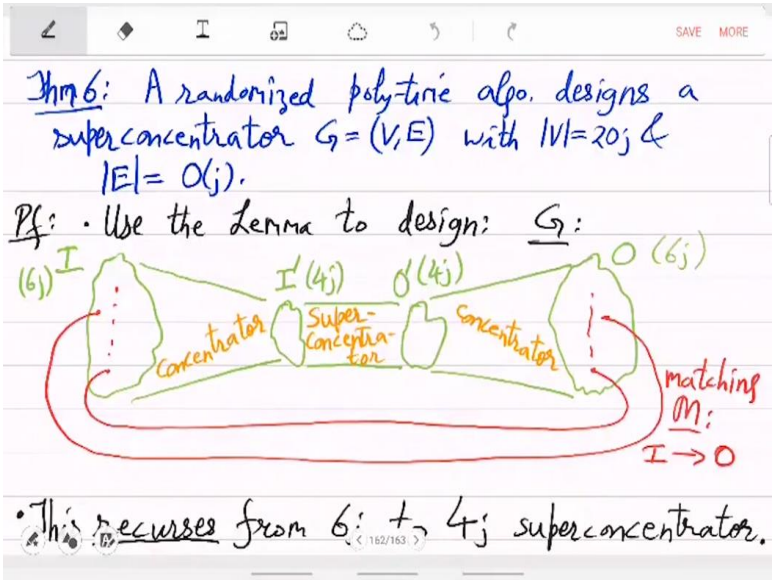
Probability for Computer Science
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology - Kanpur

Module - 8
Lecture - 31
Streaming Algorithms I

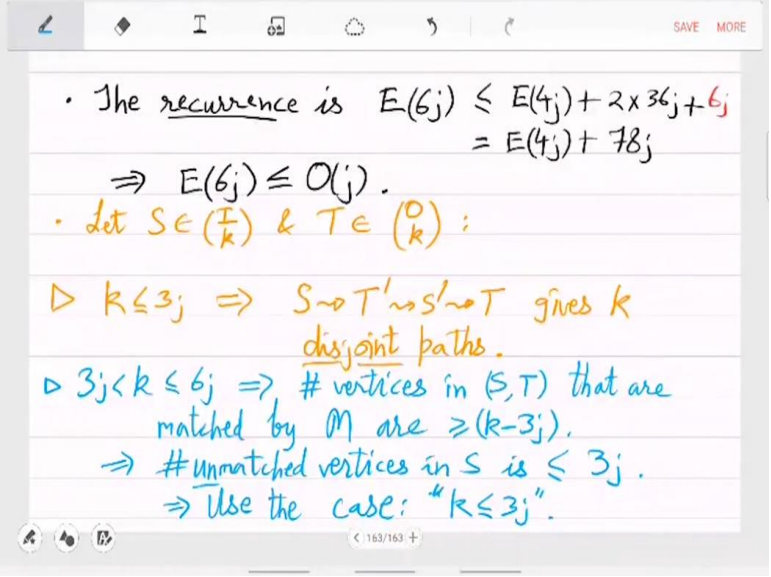
So, we are at the end of this long proof, this major application of probabilistic methods to show that a super concentrator exists of linear size, and it is an infinite family. So, what you do is, these green things will be known, and then we connect them.

(Refer Slide Time: 00:33)



So, connect I with I prime, this 6j to 4j concentrator, and a copy of this reversed O prime to O. And in the middle, I prime to O prime, this 4j to 4j, use a super concentrator. So, what this is doing is, it is reducing the 6j demand of super concentrator to 4j. It is a recursion to two-third. And then for some technical reason, we also need new edges, direct edges from I to O. So, I to O, there are these edges that we are drawing, which we call matching, just a bijection. So, identify with every vertex in I, a unique vertex in O. So, you have this recursion from 6j to 4j. And let us just finish this proof.

(Refer Slide Time: 01:46)



So, the recurrence is the number of edges needed for 6j concentrator; this is number of edges needed for 4j concentrator plus how many more edges are needed? The concentrator; so, that is 36j times 2 plus the matching ones, which will be another 6j. So, this is equal to 4j + 78j. Now, in this recurrence, 6j is basically being two-third to 4j. So, the number of times you will make recursive calls, the depth of the tree; recursion tree is around log j.

But more importantly, since 78j is being added each time, so, it is j + j by 2 + j by 4 kind of a geometric progression. So, what you get is that, linear in j. So, the number of edges overall is only linear in j. And finally, why is this thing a super concentrator? So, let S be in I chose k and T be in O choose k. So, you take arbitrary S and T; S in the input nodes, T in the output nodes, of any size up to 6j, of course.

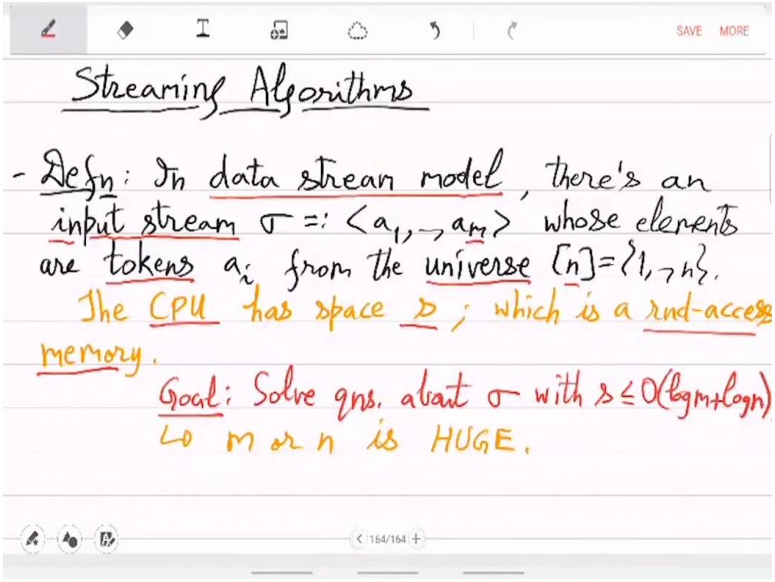
So, if k is; either k is small, smaller than 3j; then what happens? Then the concentrator property kicks in. The concentrator property says that for this S, there is a T prime in I prime. And similarly, for T, there is an S prime in O prime. And then, for S prime and T prime, super concentrator property kicks in. So, you get that S to a T prime, T prime to an S prime and S prime to T; there are k disjoint paths.

So, S to T prime comes from concentrator, S prime to T comes from concentrator, while T prime to S prime comes from super concentrator. So, that is for small k, but there is another case that of k more than 3j; but then, it will be below 6j. What about this case? So, in this case, concentrator will not help, because concentrator only can see up to 3j; it is a 6j, 4j, 3j concentrator. For this, we will actually use the matching edges.

So, what you do is, you note that, since the k is more than half of I, so, you can actually go from some vertices of S already matched to some vertices of T. So, number of vertices in S, T that are unmatched by M. So, for example, if k is 3j + 1; 3j + 1 means that at least 1 vertex will be matched. So, you actually get number of vertices that are matched is at least k - 3j, which means that number of unmatched vertices in S is at most 3j.

So, there are actually only maximum 3j vertices that we have to take care of, because they are unmatched; so, you cannot use the red edge, but 3j is fine. So, now you use the concentrator. So, for them, you can use the case k less than equal to 3j. So, that will give you k disjoint paths overall from S to T. So, this finishes the proof. So, once you have concentrator, you also have super concentrator by this construction. And it is linear sized in terms of j, it is an infinite family and this can be produced or generated by a randomised polynomial time algorithm.

(Refer Slide Time: 08:03)



So, this finishes the major topic of probabilistic methods. You have seen a number of examples by now. Hopefully this gives you a good taste of probability in computer science and combinatorics. Let us now move more into computation and more into practical computation. So, that will be streaming algorithms. So, streaming is exactly what it says, which is data streaming over the internet. So, for example, streaming videos.

So, when you are watching a movie, it is a very big file, it is not downloaded immediately; it comes bit by bit, it comes in batches and maybe your computer also will not store it completely. It will only store it bit by bit or frame by frame and just show you the movie

second by second basically. So, what you have to remember is that the working memory is much less than the overall data that was seen.

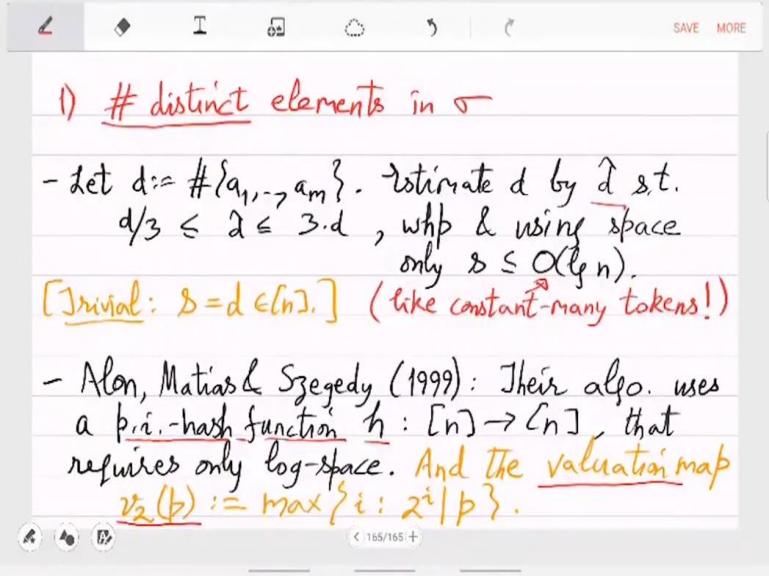
And the practical questions here are very interesting and they necessarily require probabilistic methods. So, let us make it mathematical. So, in data stream model, there is an input stream; we will call it sigma. It is a very long sequence of bits that you cannot completely store. So, you will store it only in small chunks. So, a 1 to a m are the bits in the stream; m is very large, whose elements are tokens a i from the universe, let us say n.

So, in the data stream model, you have an input stream that has tokens coming from a universe. So, this m and n, think of it as very large. And the machine which is getting the stream, the CPU has space only s; which is a random access memory. So, the memory is limited by s; it will be much smaller than n and m, in fact exponentially smaller, and it is random access.

So, it can just decide to randomly pick within the chunk of the data stream, and then, also do randomised computations. And the goal is, solve questions about sigma with s less than equal to log of m and n. So, in logarithmic space essentially, you have to solve questions about the stream. So, this is the goal which also means that m or n is huge. So, you cannot really store it, you cannot remember all the tokens in the data stream.

So, you will only remember few things; and then, from that, you will want to solve the questions. For example, how long was the data stream? That is one question. Or how many distinct elements were there in the data stream? Whether there was an element whose frequency was very high compared to others? So, those questions you can ask. And they are actually required in practice; the solution is needed. So, we have time only for 2 questions.

(Refer Slide Time: 13:36)



So, first question we will solve is, what is the number of distinct elements? Computing the number of distinct elements you saw in this very long stream. So, let d be that number. So, by the end of the day, you have seen a 1 to a m, all of them, but you are only interested in the distinct ones. How many distinct ones did you see? So, estimate d by d hat such that, let us say d hat is neither too big nor too small. So, let us say it is between d by 3 and 3 d.

Note that it seems very hard to exactly calculate d, because you have only log n plus log m space. So, in that, you cannot really store all the distinct elements. So, you will do some clever tricks by which you will approximate d. So, this is the approximation you want, d hat. And it will be probabilistic; so, with high probability and using space only s less than equal to log n.

So, storing exponentially fewer; in fact, this is like storing only constantly many tokens, because storing 1 token will already require log n space. And if you only have constant times log n space, this is like storing only constant many tokens. So, by just storing constant many tokens at any given point of time, you are still able to approximate their number. So, that is a big goal; that is a big question.

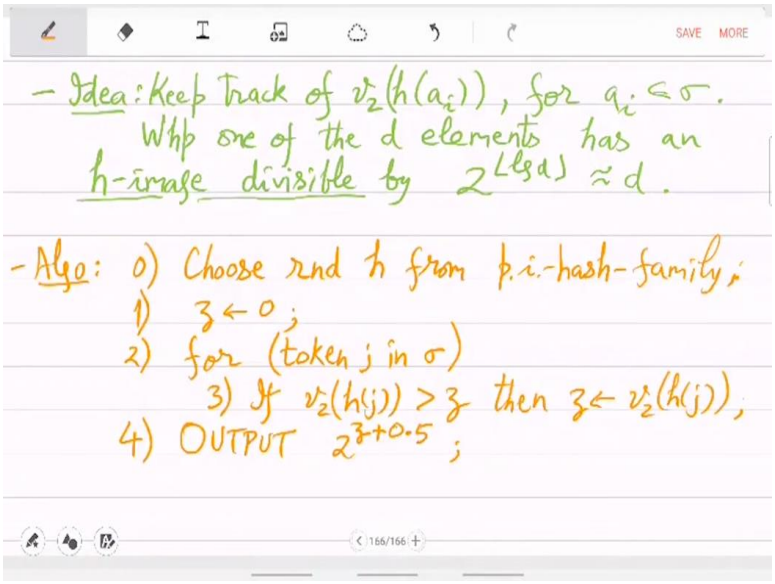
Of course, trivial is s = d; this is trivial. So, you just keep storing whenever you see a new token, but then, that will be like d, which is like in the worst case it is n. So, in the worst case, space required will be n, in the trivial algorithm, and you want to improve it exponentially. So, we will now give a surprising algorithm for solving this problem; very practical

algorithm; so, which was given by; the version we will see is by Alon, Matias and Szegedy, from 1999.

So, their algorithm uses a hash function, so, pairwise independent hash function h, which is from the set of tokens to themselves. So, basically, think of the hash function as a map which you can efficiently compute. So, you can efficiently store also, you do not need space n for that; it is a compactly presented function that requires only log space. And it also needs a map, the valuation map, v 2.

So, v 2 for a number is essentially the highest power of 2 that divides it. So, it is the maximum I, such that 2 raised to i divides p. So, just remember this. We will work with the hash function which is random from a pairwise independent hash family, and the valuation map which just tells you what is the highest power of 2 dividing your number. So, what this algorithm does is, first it applies the hash function on the token, and then it computes the valuation. It is that simple.

(Refer Slide Time: 19:54)



So, the idea is, keep track of the valuation of the token that you are seeing, where A i is the ith element in the stream. So, keep computing this hash function and its valuation. So, hash function actually gives you a number in the image. The number A i is basically going to a number which is image of A i. And so, you can compute the valuation. And this valuation, you want the maximum one.

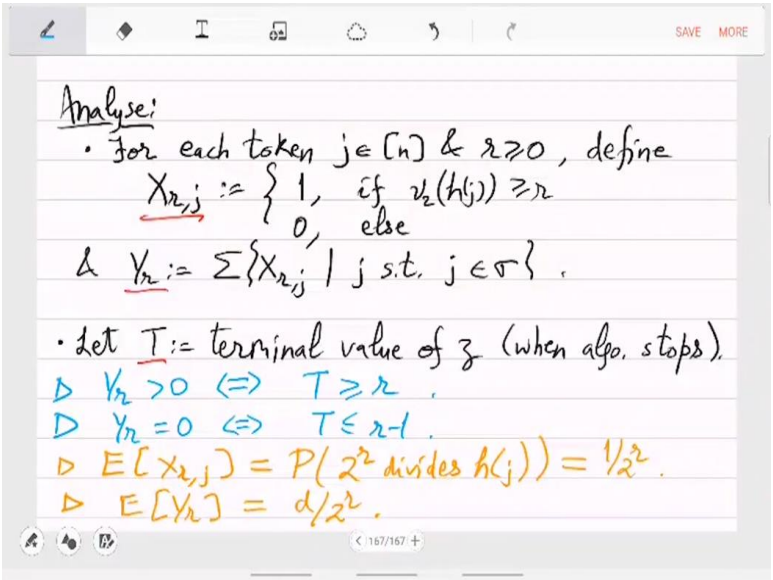
So, the idea basically is, the importance of this is that, with high probability, one of the d elements has an h-image divisible by 2 raised to log d. This is the idea, that since there are d elements, so, one of them; since we have picked h in a random way, one of the image will be a number that is divisible by many powers of 2. How many? log d many. This is the vague feeling based on which the algorithm will be designed. And 2 log d is roughly d.

So, in other words, if you look at the maximum valuation of hash image, that should give you around the number of distinct elements d. That is the idea. So, let us implement this as follows. So, choose a random h from pairwise independent hash family and initialise a variable z. Now, let us look at the stream. So, when you are looking at the jth token, if the valuation of this token exceeds your current information which is z, then you need to update your information.

So, z is now this maximum valuation that we have found till now. So, keep doing this. So, this is doing nothing but looking at the data stream and just updating to the maximum valuation seed of the tokens. So, that is stored in z. Ultimately, output 2 raised to z. So, by the intuition, by the idea, you should output 2 raised to z, but maybe this may not give a good error probability. So, let us increase it slightly.

So, output maybe an overestimate, z + 0.5. So, this is 2 power that. So, the algorithm is really based on the vague idea sketched above. But does this make sense? Can you give guarantees? How good is this algorithm? So, let us do that analysis which is important.

(Refer Slide Time: 24:42)



And there you see the beauty of probability, which you have already seen when you showed the existence of pairwise independent hash family, and this is just another application. So, for each token j, define a random variable X r j, which basically tries to capture whether the valuation came out to be r or not, valuation is at least r or not for the jth token. So, it is 1 if the valuation of this jth token is at least r, 0 otherwise. And define Y r to be the sum of this.

So, j such that a j appears in the stream. So, for all the tokens, in fact here we are talking about; so, this I think should be called then m, for each token j 1 to m. No, sorry, maybe not. Hash function, we are applying on j. So, then I should maybe put j here. Yeah, j appeared in the stream. So, for all these tokens, j which appeared in the stream, they may appear many times, but at least once they appeared.

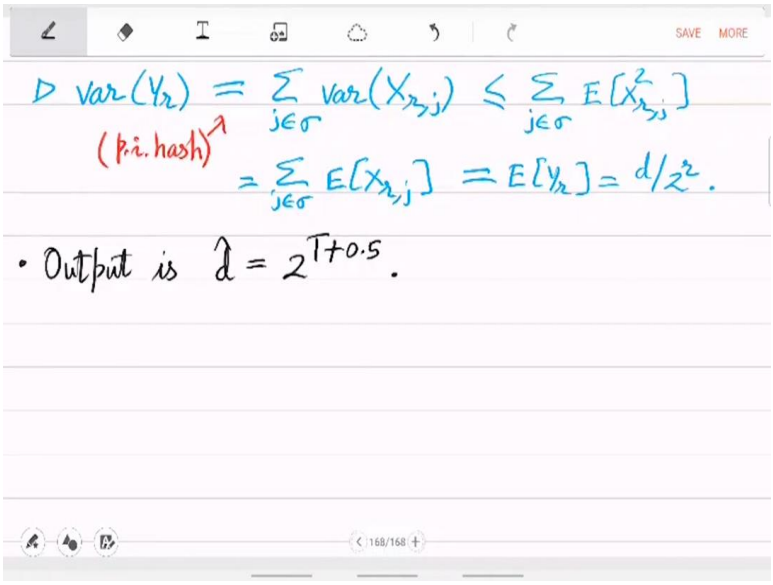
We are defining these random variables X r, j which is to capture whether the hash function image of j has a valuation that is at least r. And Y r then means that whether there was some token whose hash value is divisible by 2 raised to r or more. And let T be the terminal value of z, when the algo stops. That is T, what was the maximum that z reached. So, now, some sequence of some simple properties. So, first property is that Y r positive.

This means that there is some j with high valuation. So, the terminal value will be that, at least r. This also means that if Y r is 0, then T is r - 1 or less. Something about the expectation; so, what is the expectation of X r j, this indicator variable? So, the random thing here is hash function h. So, as you change it, what is the chance that valuation is at least r. So, that is the probability that 2 raised to r divides h of j.

So, what is the chance that on this j which is fixed in this argument and hash function is randomly picked one; 2 raised to r divides it. That is, 1 over 2 raised to r, because h j; so, this comes from the first property of hash function, that a j basically takes random values; from that; and which means that expectation of Y r is d by 2 raised to r. So, we have good control on the expectation.

So, expectation of Y r is; that really depends on the number of distinct elements. And then to give error probability of the algorithm, we will also need variance, because we will need concentration bounds.

(Refer Slide Time: 30:27)



So, what is the variance of Y r? Now, because of the pairwise independence, we have linearity of variance. So, this is really because of pairwise independent hash. So, variance of Y r will be understood if you know variance of X r j, which by definition is, if you recall expectation of X square minus expectation of X whole square, so, it cannot exceed expectation of X square.

X square is an indicator variable; so, that is the same as expectation of the variable itself, which you understand. That is the same as expectation of Y r. So, we calculated it, d by 2 raised to r. So, we understand both expectation of Y r and variance. And so, we can use a concentration inequality. So, before that, let me look at the outputs. Output is d hat; it is T + point 0.5; maximum z, that is T. So, d hat is 2 raised to T + 0.5. And now, based on the expectation and variance, we will see, can d hat be very small or very large?