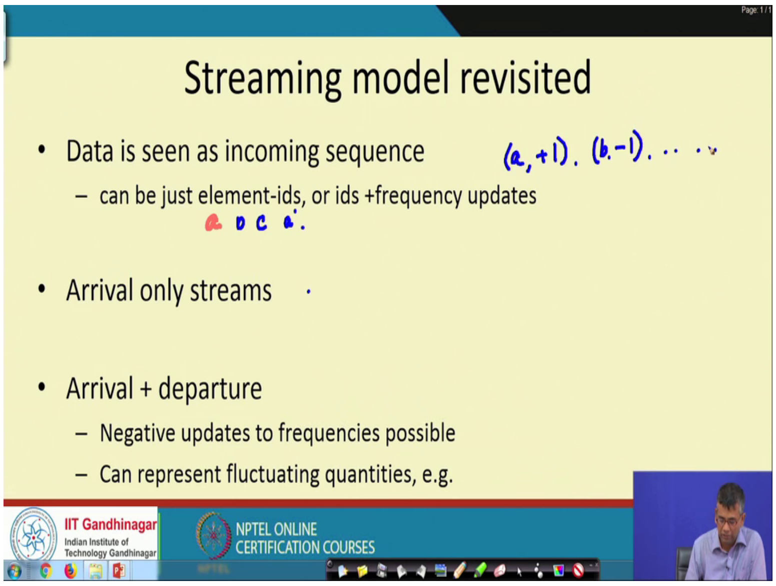
Scalable Data Science
Prof. Anirban Dasgupta
Department of Computer Science and Engineering
Indian Institute of Technology, Gandhinagar

Lecture – 09
Frequent Elements

Hello, everybody. Welcome, to the lecture of Scalable Data Science. Today's lecture is on Frequent Elements.

(Refer Slide Time: 00:28)



Streaming model revisited

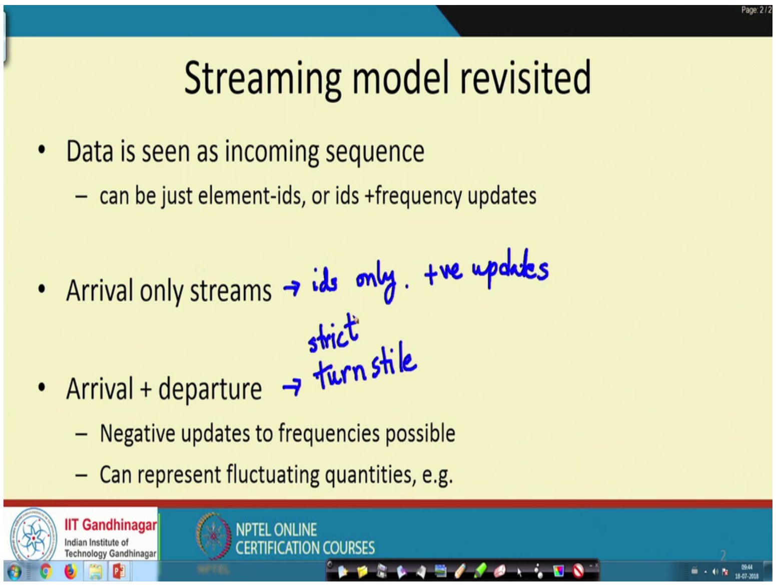
- Data is seen as incoming sequence $(a,+1).(b,-1).\cdots$
  - can be just element-ids, or ids +frequency updates $a\ b\ c\ a'.$
- Arrival only streams
- Arrival + departure
  - Negative updates to frequencies possible
  - Can represent fluctuating quantities, e.g.

IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES

So, before we start the lecture we will just revisit the streaming model that we have been talking about. In this model, we are seeing data as an incoming sequence alright and this sequence could either be element ids. For instance they could say that we are getting id a we are getting id a or id b or id c, id a prime and so, on or you could have a model in which you say that instead of just getting the id a we get the id a along with and update to the frequency of id a.

We get the id b and along the along with that and update to the frequency of id b and so on, right. These are two models that we sometimes using the changeably as you can imagine that the second model the update model is a little more powerful right. So, there are two types of streams that we typically consider. The one is arrival on the stream right.

(Refer Slide Time: 01:31)



Streaming model revisited

- Data is seen as incoming sequence
  - can be just element-ids, or ids +frequency updates
- Arrival only streams → ids only . +ve updates
- Arrival + departure → strict turnstile
  - Negative updates to frequencies possible
  - Can represent fluctuating quantities, e.g.

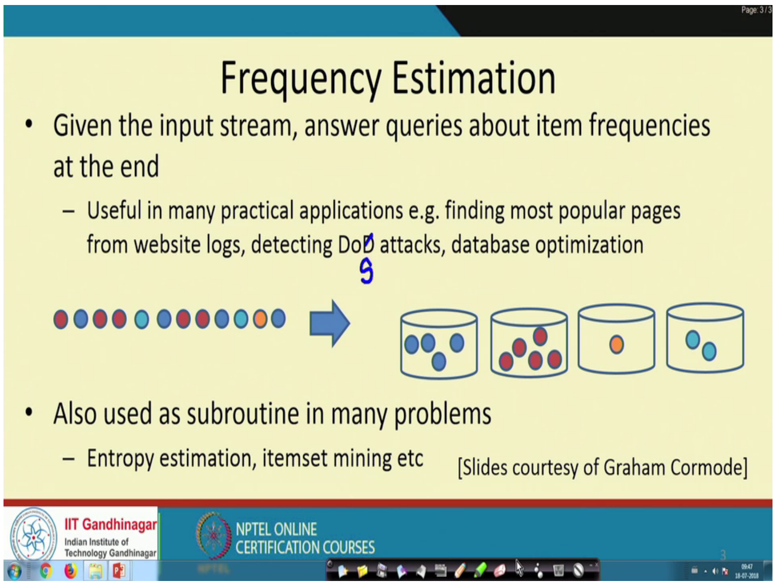
What that means, is that either you are getting the ids only which effectively means that the update frequencies are one every time, right or if you are getting updates you are getting positive updates always or at least non-negative updates, that is your never facing a situation when you are having to delete an element.

The next I mean the most powerful model; however, is when you can combine both arrivals, and departures and such models can arise for instance if you are measuring fluctuating quantities, right. So, imagine that the price of a stock right you sort of at every day or every second you might be getting an update to the current price of the stock, right and this and this could be. So, your stream could look like an idea of a stock and update to the price right and then this update is can be both positive and negative of course.

So, the so, this is also known as the turnstile model sometimes because yeah the picture that you have in mind is that you have the turnstile in front of the through which people enter the metro stations or maybe sometimes the train stations. So, sometimes people could be going in the turnstile turns right sometimes could be going out the turnstile turns to the left, right. In most cases in most cases we will deal only with streams when deletions can be made to objects that you have already inserted, right which means that the effective total frequency of any particular object or item can never go below 0, ok.

So, this is also sometimes called the strict turnstile model. However, this is the typical cases that will we dealing with always this. So, unless I sort of say otherwise we are always in the I mean whenever I say arrival plus departure we are always in a strict arrival plus departure model in which you are getting both positive and negative updates. And the, but the negative updates can never result in the total frequency of any item being less than 0, right. So, most of our work will be in this lecture yeah in will be in the arrival only streams. We will touch upon the arrival plus departure streams in the next lecture.

(Refer Slide Time: 04:18)



Frequency Estimation

- Given the input stream, answer queries about item frequencies at the end
  - Useful in many practical applications e.g. finding most popular pages from website logs, detecting DoD attacks, database optimization
- Also used as subroutine in many problems
  - Entropy estimation, itemset mining etc

[Slides courtesy of Graham Cormode]

S
IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES

So, the problem that we tackle in these two lectures today's lecture and the next strand the next lecture is the question of estimating frequencies right. So, in some sense so, here is a question, that suppose imagine this is a stream you get the red ball, you get the blue ball, you get two red balls, you get a turquoise ball, you get a blue ball and so on, right. And, at the end you have to answer the question that given a particular color how many balls have you seen for that color, right.

For instance if the if I give you the color turquoise you have to say that I have seen 2 balls of this color or red I have seen 5 balls of this color, right. So, in some sense you have to you have to create this histogram right which I have sort of pictured as a bunch of buckets, right.

So, why is this a useful question first of all. So, this I mean as you can imagine this is obviously, a very basic question that you need to answer in most data analysis tasks. A very sort of common use is when again bringing back our example of network switches. Suppose, you are you are designing some basic analytics tasks to be run on network switches and one of the most common things that you could try be go try to detect is whether there is a denial of service attacks that is happening whether there is a denial of service attacks that is happening.

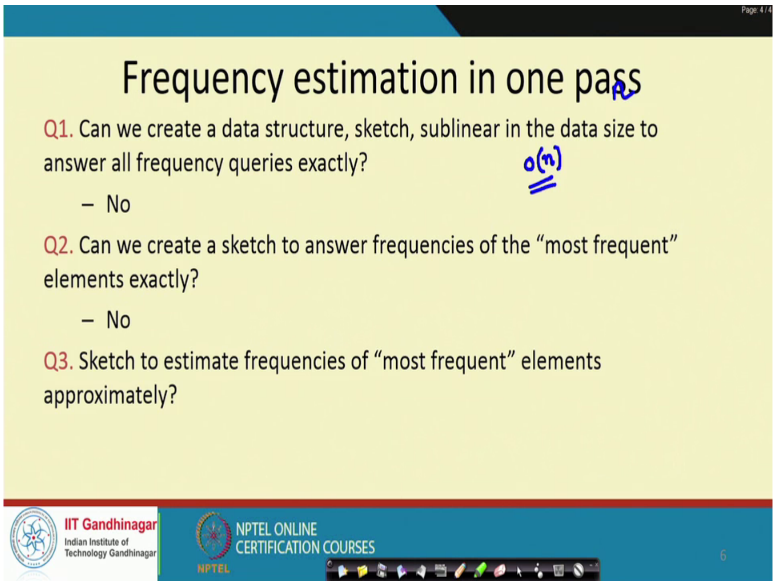
So and how would I detect that? Well, we could always track which are the IPs that are getting the most number of packets, right, in maybe in the last few seconds maybe in the last 1 hour, right and that good form part of our machine learning pipeline or data analysis pipeline that goes to predict that there is potentially a denial of service attack that is why delta, right. Another common use case is in database optimization. So, as you might have as you might already know doing I mean if you have a bunch of complicated joints that are doing in databases, right.

Sometimes it makes sense to get an estimate of the various relation sizes after each of these joints, right, so that you can optimize this joint pipeline and if the data if the databases are huge. And, and they are stored in the in disks or in tape drives then it is it is very useful if you if you can do one pass over each of these data sets over these data sets and then get some estimate of the frequency of different of different element types of the of the different elements,.

You could also imagine that if you are a huge web company, right. Imagine your Google or Yahoo or Microsoft, right and you are trying to find, I mean from msn dot com which are the most popular pages that are being clicked. So, you look at the website logs, right well, those machines they do not have a lot of memory to devote to your analytics tasks. So, you have to do things in very small memory, but you could try sort of building the kind of data structures that filter that will talk about in order to find out that for today’s visits these 10 websites that are linked from msn dot com are the most popular web pages or other web pages on which the Microsoft visitors clicked on the most.

Also, it is not a surprise that that these questions are used in subroutine in many other problems. For instance in stp here in estimating the entropy of a data of a data stream right in doing frequent item set mining and so on by the way the slides that you see today are courtesy of are courtesy of Graham Cormode and there is a link to the two set of slides and the lecture notes at the end of the class.

(Refer Slide Time: 08:20)



Frequency estimation in one pass

Q1. Can we create a data structure, sketch, sublinear in the data size to answer all frequency queries exactly? $o(n)$

– No

Q2. Can we create a sketch to answer frequencies of the "most frequent" elements exactly?

– No

Q3. Sketch to estimate frequencies of "most frequent" elements approximately?

So, let us see what are the are literally formalized what are the questions that we can try to ask about this frequency estimation. The first question is this that what is the best I can hope for? The best I can hope for is that, can I create a data structure right let us call that a sketch for now can I create a data structure or a sketch that is sub linear in the size of the data, right because if I had to use space that is linear in the size of the data I can always use something like a I mean an array right something like an array and associate sort of counters. So, that is trivial.

So, can I use a data structure that is sub linear in the size of the data right and can answer all frequency queries accurately which means that if the if the number of unique ids is n what you want is a data set is a data structure that is small o of n, right is it possible to do this? Unfortunately not, right. It is not hard to say that I mean again using information theoretic arguments that you cannot hope to design something that is small o of n in space and that answers all the frequency queries exactly, ok. So, that is gone let us come to the next question.

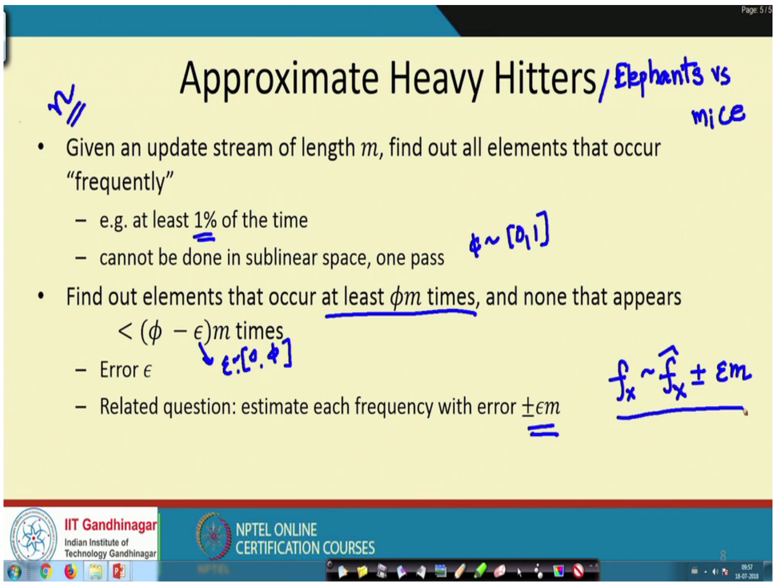
So, instead of trying to answer all frequency queries sometimes the issue is that maybe there is a huge number of tailed queries of sort of tail elements right which are very small frequencies and the question then is can I so, suppose I do not worry about them.

Suppose, I only worry about giving accurate answers to the most frequent elements ok, can I do this? So, I could certainly do this if in the offline setting, right. For instance if there are only let us say 10 frequent elements, 10 elements which are most frequent and am not defining this most frequent very accurately right now very precisely right now. Suppose, there are 10 elements which are high frequencies everything else is low frequency.

So, in sort of one pass over the data, right I could maybe try to guess one of these elements right or basically if it is offline then I could always sort right and then and then and then and then find out which are the most frequent elements and then only show the frequencies of this right. However, we cannot really do this again right as you already see from my answer because once you sort once you try to sort it you are already using linear space, right. So, even this is not really doable in sub linear space right. So, when I say not really doable it is a combination of sub linear space and one pass ok.

So, in one pass sub linear space the first two questions that we are trying to answer are not possible, ok. Let us come to the third question which is a relaxed version of the first two. Can I try to approximate the frequencies of the most frequent elements, if I cannot answer them exactly can I try to give the answers approximately and am not even making precise what approximation really means, right. So, we will come to this and it turns out that yes for certain definitions of this approximation it is possible to do this right and even that answer to be a non trivial question to study which is and that is what will study in this lecture and the next one.

(Refer Slide Time: 12:10)



Approximate Heavy Hitters/Elephants vs mice

- Given an update stream of length m, find out all elements that occur "frequently"
  - e.g. at least 1% of the time
  - cannot be done in sublinear space, one pass
- Find out elements that occur at least ϕm times, and none that appears $< (\phi - \epsilon)m$ times
  - Error ϵ
  - Related question: estimate each frequency with error $\pm\epsilon m$

$\phi \sim [0,1]$

$\epsilon : [0, \phi]$

$f_x \sim \hat{f}_x \pm \epsilon m$

So, let us try to define what we mean by approximation. So, by the way this is this particular problem the question of finding out the frequency of the most frequent elements has various names in literature. It is called heavy hitters, it is called it is called elephants versus mice and so on and they all point really are talking about the same problem. So, let us see let us try to define it this way that, suppose we have an update stream of length m. Remember that we are always talking about and at this point we are talking about a universe whose size is n. So, there are n distinct elements well we could even do a little better we could say that the universe is big, but the number of distinct elements that you see is n.

So, n elements right, but the length of the stream m is bigger than n right because an element can come multiple times. So, given an update stream of length m, find out all elements that occurred frequently. So, let us say by frequently I mean that it occurs at least 1 percent of the time, which means that their frequencies is at least 1 percent times m right 0.01 times m, unfortunately this cannot be done in sub linear space one does.

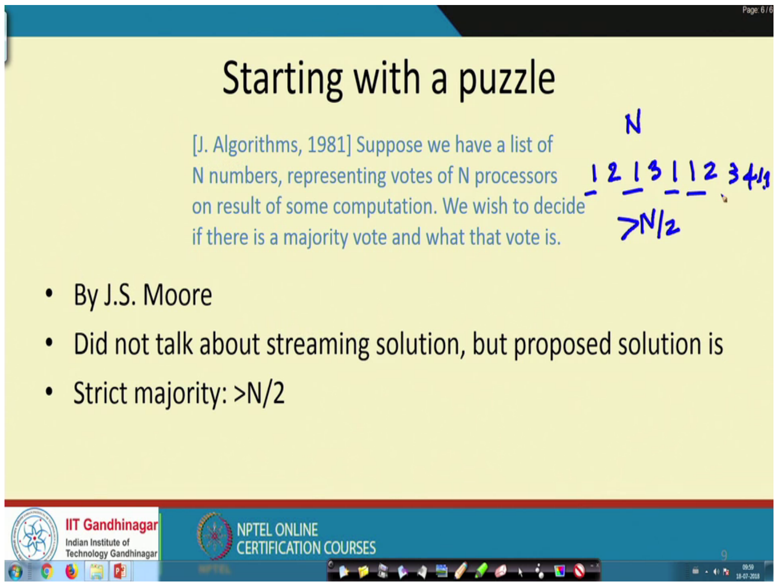
So, you cannot find out the set of all elements that the exact set of all elements that appear 0.01, I mean 0.01 m times right in space that is that is that is sub linear, right and the and the while am not giving in the proof the intuition is this that when you see an element the first time you have no clue of whether it is going to be it is going to become one of these heavy hitters or not right or one of these most frequent elements are not right and you have to keep it. And therefore, you have to keep around this elements in order to it because if you do not keep in keep it around then you are never going to I mean then you miss it is frequency, right. So, it is it is possible to design possible to design sort of I mean formalize intuition into a complete proof and this was a question to that via dust.

So, the version of question 3, the formal version of question 3 that will now try to tackle is this that, suppose can I do this? Right, that can I give a guarantee that says that we will find that all elements that occur at least some phi times m. So, this phi was 1 percent. So, think of phi as some number between 0 and 1, ok. So, what I want to guarantee is an algorithm what am going to give is an algorithm that guarantees is that all elements that occur at least phi times at least phi m times are there and none that appears less than phi minus epsilon m times are there for epsilon that is between 0 and phi, ok. So, epsilon is error.

So, remember, so, see that there is a gap here while you are guaranteeing that that all elements at least at least phi m frequencies at least phi m will be there, right. I am also guaranteeing that all elements those frequencies less than phi minus epsilon will not be there for the things in the middle for the elements whose frequencies are between phi m and phi minus epsilon m am not guaranteeing anything, ok. So, this gives me the slap to work with. So, epsilon is the error a very related question and one that will actually tackle in this in this particular in this particular talk is to estimate each frequency right with error plus minus epsilon m for some for some other epsilon.

So, do not think of this epsilon and this epsilon. Well, it could it could think of these epsilons as being the same right what will say is that if the if the actual frequency of an element x is f x will give you some f hat x such that f x is not too different f at x and f x are not two different, right which means that they somehow differ in as plus minus epsilon m, right. And this statement again is not formal the exact form of the statement will depend on the on the algorithms that we on the particular algorithm that will give, but this is more or less what we want to do, and this is the version of the question that will actually tackle.

(Refer Slide Time: 17:06)


Starting with a puzzle

[J. Algorithms, 1981] Suppose we have a list of N numbers, representing votes of N processors on result of some computation. We wish to decide if there is a majority vote and what that vote is.

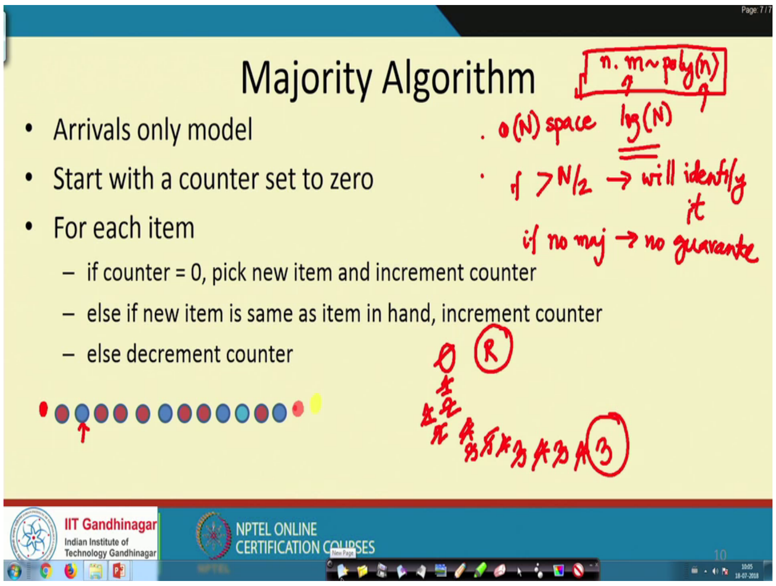
- By J.S. Moore
- Did not talk about streaming solution, but proposed solution is
- Strict majority: >N/2

So, let us start with the puzzle before we give the algorithms that suppose we have a set of n numbers right and this puzzle was supposed by J. S. Moore in 1991 in the journal of algorithms and he posted in the setting of our of our distributed computing, but it is valid nonetheless you know in other settings also. Suppose, you have a list of N numbers and these N numbers represent the votes of N processors on the result of some computation right. So, maybe the processors they have had slightly different inputs they have had slightly different randomness and they are all saying and they are all giving a vote on the on the result of this competition, right and let us say the votes are a some a some a some is a particular number.

So, let us say the vote is number 1, number 2, number 1, number 3, number 1, number 1, number 2, number 3, number 4, right and these are the votes right. So, now, he is asking that is there a majority vote, right is there a strict majority vote that is if there are if there are N processors is there a particular option that appears strictly N by 2 times, ok. Here for instance you could potentially say that maybe if I put in a few more ones the option one appears bigger than N by 2 times, ok.

So, this is easy to do as such right because you just you can just keep you can just keep a counter and so on, but again can we do this can we answer this question in sub linear in N space. So, it turns out that he is setting the setting that the puzzle was asked him did not really sort of talk about streaming because streaming was not formally defined as a model as of then, but the solution that he proposed is actually a streaming algorithm. It is actually streaming algorithm that uses constant or logarithmic amount of space, right. So, it is pretty interesting and let us look at the answer to this puzzle.

(Refer Slide Time: 19:16)



Majority Algorithm

- Arrivals only model
- Start with a counter set to zero
- For each item
  - if counter = 0, pick new item and increment counter
  - else if new item is same as item in hand, increment counter
  - else decrement counter

$\boxed{n.\ m \sim poly(n)}$
. o(N) space $\underline{\underline{\log(N)}}$
. if $> N/2 \rightarrow$ will identify it
if no maj $\rightarrow$ no guarantee
Ⓡ
③
Page: 7/7
IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES

So, here is what we want to do, right. Again, I am not giving the exact and I mean am sort of casting the puzzle in our streaming model where we are talking about arrivals only. So, suppose we have this stream red, blue, blue, red, blue, red, red, red, red, blue, red, red, yellow I mean there is no yellow, turquoise, blue, blue maybe I should put in a few more colors maybe I should have a pink out there maybe I should have a yellow out there and so on. So, now the question is that is there is there a is there a just to make sure that there is a strict majority am putting in a few more reds, so.

So, now, the question is there is there a sub linear solution to the question of trying to see trying to pick a majority item, right. So, here is what we and what I want to guarantee right I want to give you an algorithm right that is going to have the following guarantee that if that it is going to use only small o of N space, right. In fact, it is going to use only logarithmic in N space right. So, just aside I mean typically what we assume is that n and the length of the stream m they are related as polynomials of each other, right.

So, when I say logarithmic in n versus logarithmic in m basically the same thing right with the difference of constants and it is useful to remember this because, I will keep on interchanging sometimes it is useful to state it in terms of logarithmic in m sometimes it is useful to state the boundary in terms a logarithmic in n, but I mean that is the setting that we mostly deal with, ok. So, coming back to the problem right so, can I give an algorithm that uses log N space, right and that has the following guarantee that if there is a majority counter that if there is a majority item you will have it in hand will identify it, right. However, if there is no majority item, right and this is the slack if no majority item, right then the algorithm is not giving any guarantees no guarantee.

In fact, then the then the sorry then the item that it that it has at hand it will have at hand might be a completely random item might be a pretty bad item ok. So, there is nothing we can say about this ok. So, but of course, if we were allowed to do one more pass over the data right and this we are going to do only in one pass for the data if you what will for allowed to is one more pass of the data you can do away with this with this restriction right because once you pick an item the another pass over the data you can sort of say that ok, this item is really majority item or there is no majority you can distinguish within these two cases, right. So, in one pass we would not be able to do this.

So, so, what is algorithm, this is this sort of mysterious algorithm? It is very simple what will do is that we have a counter right a single counter that starts at 0. So, the counter will start at 0. So, for every item; so if the counter to at 0, right and you are seeing an item you pick up that item, right. So, the counter at 0 we see red. So, we pick up red, and increment the counter. So, counter becomes 1, right. Next, then we go to the next item the next item is also 0 is also red right. So, I have the item right in hand and the counter is 1. So, because the item that I see next which is red is the same as the item that I have in hand red I increment this counter, right. So, if the new item is the same as item in hand I increment the counter.

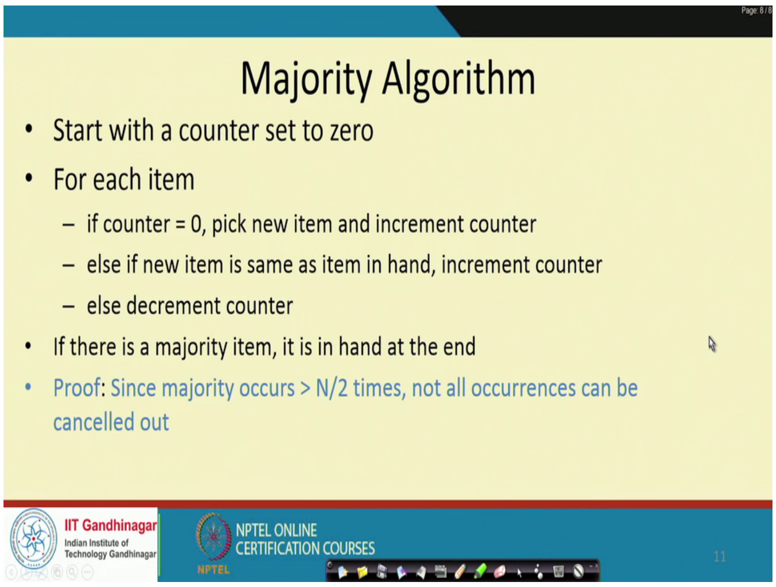
So, I always have two things in my hand and an actual item id as well as a counter value,, but then I see blue the next element and this blue is not the same as the red. So, what do I do there? So, I do not satisfy the first if I do not satisfy the second else. So, I come to the last else which means the right decrement the counter. So, if the item that I see is not the same as the item that I have in hand I decrement the counter. So, this becomes 1, but the I, but then I throw away the blue then I get a red.

So, then I increment it to 2 then increment it to 3 4. So, it gets incremented to 4, then I see a blue it gets decremented 2 gets decremented to 3, then I, then it again increases by

2. So, it becomes 5 I see a blue which is different from red. So, it gets decremented to 4, I see a turquoise which is different from red it becomes 3, I see a red again it becomes 4, I see a blue it becomes it becomes 3, I see a let us say a red it becomes 4 I see a yellow it becomes 3, right.

So, at the end am left with an item red as well as the counter value 3, ok. So, now, what is the guarantee that we have? So, it is pretty clear that that this is log N space, right because we only are storing two I mean two things an item id as well as a counter value. And, the counter value can at most be log of m right which as we said is a is again log of n because I mean we dealing with polynomial size dreams.

(Refer Slide Time: 25:03)



Majority Algorithm

- Start with a counter set to zero
- For each item
  - if counter = 0, pick new item and increment counter
  - else if new item is same as item in hand, increment counter
  - else decrement counter
- If there is a majority item, it is in hand at the end
- Proof: Since majority occurs > N/2 times, not all occurrences can be cancelled out

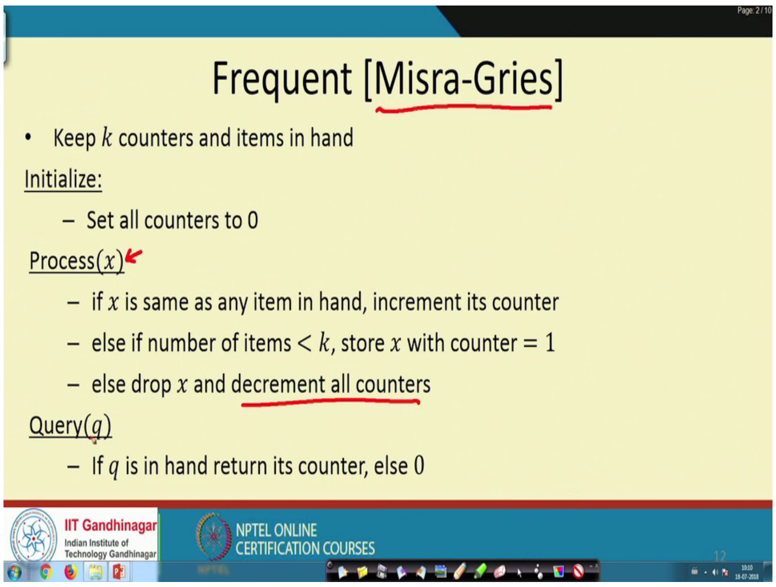
IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES
11

So, the proof is surprisingly simple right. So, here is a claim if there is a majority item it is in hand it is there with the algorithm at the end, right. So, the so, the first step is very important if there is a majority item we have it in hand am not guarantee you anything if there is no majority item and why it is true it is really simple, right. So, think of when are we dropping a majority item, right. So, when you sort of when you delete when you throw away an item, right so, remember at some point we are at some I mean if the counter value is 0, we are dropping the item that we in hand and we pick up the new item right. If I decrement the counter that is the same as dropping one copy of the item, right.

So, now, how many times can you drop a majority item, ok. So, each drop of the majority item happens because of one other occurrence of some other item, right because the only time that we that we that we drop an item is when we get an item another item and that item does not match the item that I have in hand. So, each drop of majority item called I mean can be charged to the arrival of one other copy of another item right, but the length of the stream is capital N and there are capital n by greater than capital N by two majority items.

So, the number of other occurrences of other items is strictly less than N by 2. So, therefore, you cannot account for all the I mean all the majority items being dropped all the copies of all the majority items being dropped, right. Because less than N by 2 arrivals because less than N by 2 arrivals cannot account for greater than N by 2 copies of the of the a majority item. So, that is it really ok.

(Refer Slide Time: 26:58)



Frequent [Misra-Gries]

- Keep k counters and items in hand

Initialize:

- Set all counters to 0

Process(x)

- if x is same as any item in hand, increment its counter
- else if number of items $< k$, store x with counter = 1
- else drop x and decrement all counters

Query(q)

- If q is in hand return its counter, else 0

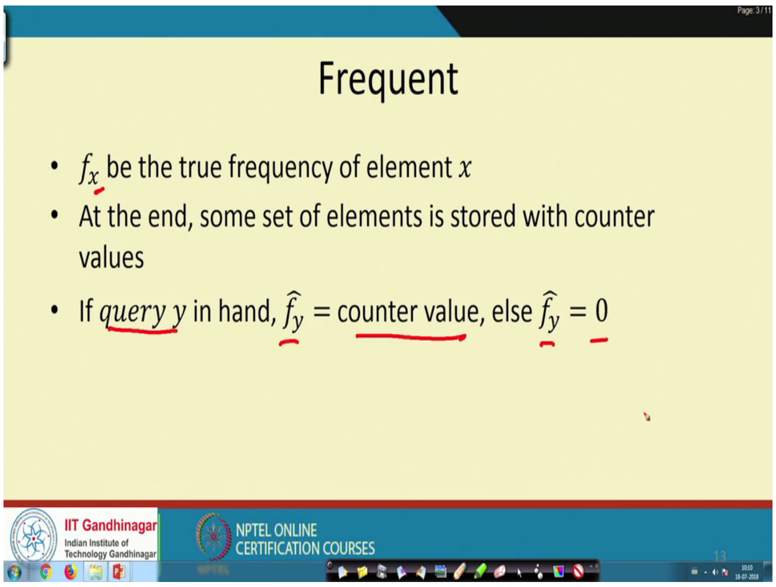
So, now, this can actually be turned into the kind of algorithms that we are looking for and this algorithm was proposed by Misra and Gries alright and, but surprisingly it has been discovered multiple times re discovered multiple times independently, right each time with a slightly different variant maybe. So, here what do we do instead of keeping one item and one counter we keep k items and k counters. We will decide how to how to set the value of this k later, right. So, we have k items and k counters we first initialize all the counters to 0 right and all the item ids are null say.

So, if you so , now, I so, now, am trying to process a particular steam a particular element of the stream x. So, if x is the same as any of the k, I any of the items that you have in hand and you have at most k items in hand in that case increment the counter of this item. So, each of the counters k counters corresponds to one of the items that you have in hand right and if x matches any of the item items that have an increment the counter, else x let us say x does not match any of the existing items.

But, if you have space if the number of items that you have in hand is less than k, right then you store x that is fairly natural you store the item x and you associate a counter with it with the value of 1, and if none of these happens to be the case this is the most interesting setting and if none of it none of the above to happens then we drop x first of all, but we do something strange, right. We decrement all the counters by 1.

So, each of the k counters is decremented by 1. So, this might seem strange to you, but this is what really makes algorithm take, ok. So, you keep on doing this, right and now finally, suppose that the other at the end of the stream somebody gives you a query q ok. So, there is only two possible cases the query is an item id either the item id something that algorithm had it had in hand at the end of the stream in that case you return the counter value associated with that item or else you return 0, because the algorithm was not tracking this item, ok.

(Refer Slide Time: 29:59)



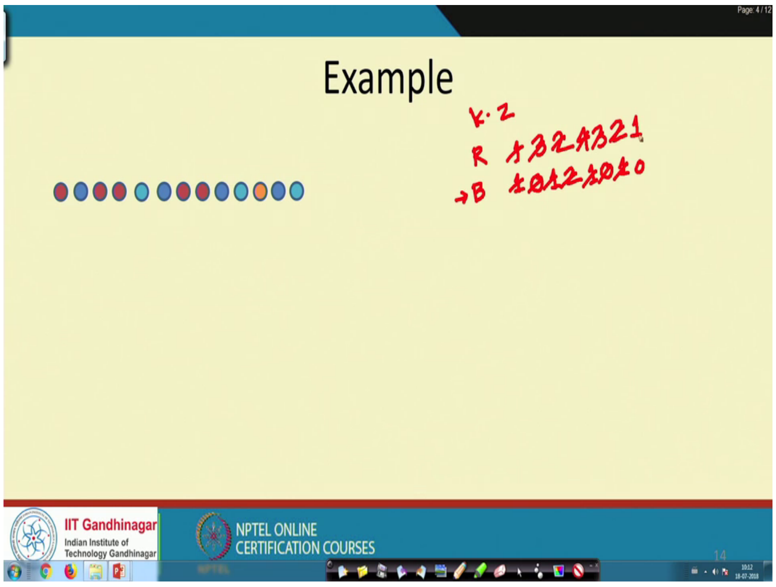
Frequent

- f_x be the true frequency of element x
- At the end, some set of elements is stored with counter values
- If query y in hand, $\widehat{f_y}$ = counter value, else $\widehat{f_y} = 0$

IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES

So, let us see so, just some notation again and I think we have already done this.

(Refer Slide Time: 30:05)



Example

k · 2

R $\cancel{1}\cancel{3}\cancel{2}\cancel{4}\cancel{3}\cancel{2}1$

→B $\cancel{1}\cancel{0}\cancel{1}\cancel{2}\cancel{1}\cancel{0}\cancel{1}0$

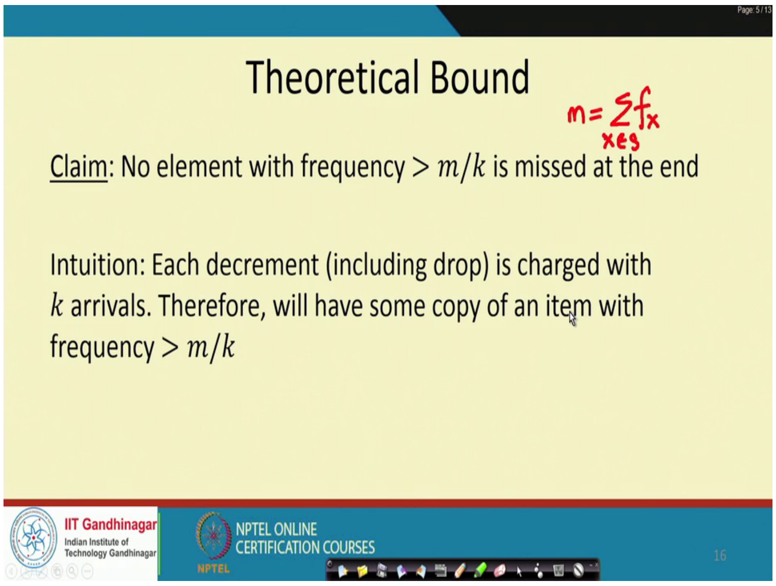
IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES

But, supposing I mean for any item x will denote it is will denote it is frequency by will denote it is frequency by f x, and at the end of the stream some of the we are storing the some set of elements or the associated counter values. If the query y is in hand then f hat of y is defined as the counter value because that is the value that you are returning else effect of y is defined to be 0, right. So, we can sort of cost algorithm in this way cost the final query in this way.

So, just let us run down a small example. So, supposing k equal to 2 right, ok. So, first you store the red with a counter value of 1, you store the blue with a counter value of 1, then red increments to counter value by to 3 increments a counter value to 3, then you can a turquoise now what do I do I do not I am only allowed to keep the two items, right and turquoise is not there. So, I dropped turquoise, but then I decrease each of them I decrease this to 0, I decrease this to 2. Then I get a blue, I increase this to one because I have blue in hand I get two reds, I increase this to 4, I get a blue this is to 2, I get again a turquoise. So, again you have to drop this, right. I decrease this by 1 and I decrease this by 1, ok. I got a yellow I decrease this to 0, right and I sorry and I decrease this to 2, then I get a blue.

So, at this point since this is 0, you can imagine the blue is not there that once a counter becomes 0 we could has will drop that corresponding item, but it turns out that in this example I get a blue again, right. So, I add blue in with a counter value of 0 and then I get a turquoise. So, this becomes 0 and this becomes one. So, that is how it is.

(Refer Slide Time: 32:08)



Theoretical Bound

$m = \sum_{x \in S} f_x$

Claim: No element with frequency $> m/k$ is missed at the end

Intuition: Each decrement (including drop) is charged with k arrivals. Therefore, will have some copy of an item with frequency $> m/k$

IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES
NPTEL
16

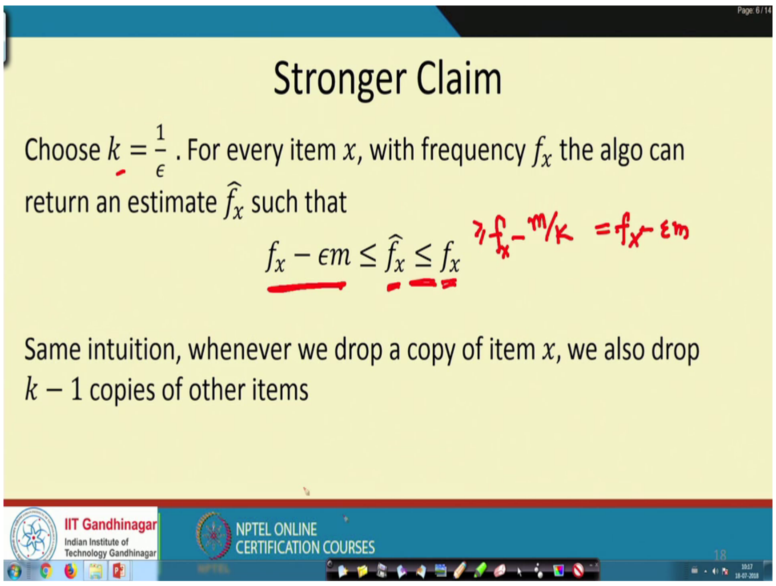
So, why is this a nice algorithm. So, here is the first claim. The first claim is somewhat weaker than the then the final claim that that we will sort of right, but it nevertheless useful to think about it, to think about a simple question first because it is similar to the majority question that we handled. What we will claim is that, no element with frequency is strictly greater than m by k it is missed at the end, right. So, notice what is m. So, m is the length of the stream which is also the sum of the frequencies of the elements sum over all x belonging to the stream, right x belonging to stream s because every item every sort of copy every id in the item the string corresponds to one of the updates one of the frequencies of the one of the elements, right.

So, it should not surprise. So, this statement should not surprise you, right, because that because argument for this is going to be exactly the charging argument that we that we made right because each decrement. So, imagine what is happening in each decrement in each decrement or a drop we not only decrement through away this one element. But we are decreasing we are decrementing other k minus 1 elements, right because when we throw away an element when we are not able to add it to the to the list of elements that we already have right. Because they are because we have k elements we drop we drop this element right, but then we decrease the count of k other elements, ok.

So, therefore, each decrement right is charged with k arrivals because these things the I mean because these counts must have correspond must correspond to some arrival of the corresponding item somewhere in the stream, ok. So, each possible drop or each possible decrement is charged with k arrivals, but how many arrivals can there be in total? There cannot be more than m arrivals because the length of the stream is m, right.

So, therefore, therefore, if the if the I mean if the reason if there is an item with frequency greater than m by k, right, I could not have dropped or decremented all the copies of this item, right because there can be at most m by k less than equal to m by k total decrements and drops per item, right. So, per item there can be less than equal to m by k decrements of drops and therefore, if the frequency of an of an element is strictly bigger than m by k I must have that at the end, ok. So, this seems like an intuition, but it is really very formal. So, and the and the statement that that we can make is which is a more precise statement it is something like this, right.

(Refer Slide Time: 35:11)



Stronger Claim

Choose $k = \frac{1}{\epsilon}$. For every item x, with frequency f_x the algo can return an estimate $\widehat{f_x}$ such that

$$\underline{f_x - \epsilon m} \le \underline{\widehat{f_x} \le f_x} \quad \ge f_x - m/k \; = f_x - \epsilon m$$

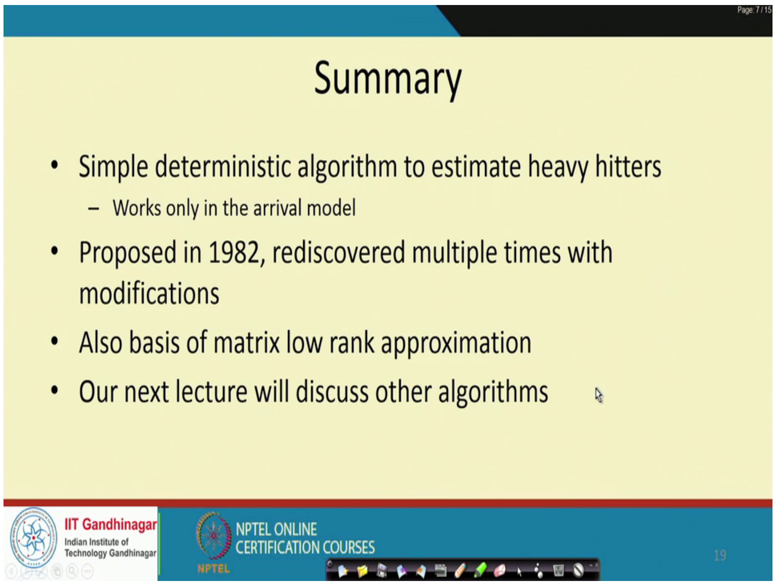
Same intuition, whenever we drop a copy of item x, we also drop $k - 1$ copies of other items

Page: 6 / 14
IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES
18

So, I did not want to I mean this statement seems mysterious, but it is really exactly the same thing that they have been that we discussed, right so. So, now, so, imagine that so, choose the value of k to be 1 by epsilon, suppose, f x is the frequency of the item x and f hat x is the frequent frequency estimate. So, you may so, remember that f hat x is either the counter value that you have in there or it is 0, ok. So, what we claim is that f x f hat x is less than equal to f x and f hat x is greater than equal to f x minus epsilon m.

So, why the former? So, the less the fact that f hat x is less than equal to f x is easy because you only sort of increase the frequency of an element when we see it in the stream. So, therefore, f hat x is at most less than equal to f x. Why is this? So, again I would not write down the formal proof of this, but the argument is exactly the same, right. So, there can be there can be at most for the element x there can be at most m by k deletions are drops, right. So, therefore, the so, the if I mean f hat x must satisfy that f hat x is bigger than equal to f x minus m by k which is equal to f x minus epsilon times m ok, again using exactly the same charging argument as before, ok.

(Refer Slide Time: 36:44)



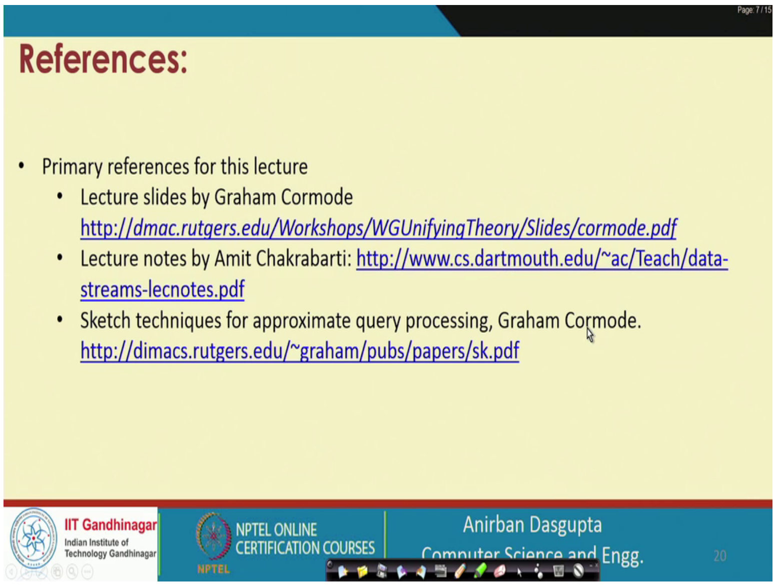
Summary

- Simple deterministic algorithm to estimate heavy hitters
  - Works only in the arrival model
- Proposed in 1982, rediscovered multiple times with modifications
- Also basis of matrix low rank approximation
- Our next lecture will discuss other algorithms

IIT Gandhinagar
Indian Institute of Technology Gandhinagar
NPTEL ONLINE CERTIFICATION COURSES
NPTEL
19

So, just to summarize I mean in this lecture we gave you a simple deterministic algorithm to estimate heavy hitters. This was this however, works only in the rival model. This was proposed in 1982, rediscovered multiple times with modifications. Very interestingly at this algorithm also forms the basis of some recent work on matrix low ranked approximations in the streaming setting. Now, next lecture will discuss a bunch of other algorithms.

(Refer Slide Time: 37:07)



References:

- Primary references for this lecture
    - Lecture slides by Graham Cormode http://dmac.rutgers.edu/Workshops/WGUnifyingTheory/Slides/cormode.pdf
    - Lecture notes by Amit Chakrabarti: http://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf
    - Sketch techniques for approximate query processing, Graham Cormode. http://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf

IIT Gandhinagar Indian Institute of Technology Gandhinagar NPTEL ONLINE CERTIFICATION COURSES Anirban Dasgupta Computer Science and Engg. 20

So, just to give you some reference. A lot of the slides that you see here in the examples that you see here are from the lecture slide this excellent lecture slides by Graham Cormode. There is also some lecture notes by Amit Chakrabarti that we have pointed out yesterday as well as a monograph by Graham that is very useful.

Thank you.