

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

Lecture - 07
Bloom Filters

Hello and welcome to our course on Scalable Data Science. Today's lecture is on Bloom Filters. I am Anirban, and I teach at IIT Gandhinagar.

(Refer Slide Time: 00:33)

Page: 23/23

Querying

ISBN present in collection?

IP seen by switch?

10.0.21.102

IIT Gandhinagar
Indian Institute of Technology Gandhinagar

NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

2

So, we are still stuck with this problem of query, we have looked at hash tables right, but again let us revisit what we have. We have we have talked about the problem of building library, where we have a lot of books, but a particular user comes to you, gives you a eyes bin or a book name, and has asked you whether this book is there in your collection and you are wondering how to answer this question efficiently. You also could be the designer of a network switch right that is up which has very limited memory. And now a particular user have again comes and asks you, have you seen this IP have you seen any traffic for this IP passing through this switch? And you have to answer this question more or less accurately at least ok. So, how do we do that?

(Refer Slide Time: 01:26)

The slide is titled "Solutions" and lists two methods for storing a set of n items from a universe U , where $n \ll |U|$.

- Universe U , but need to store a set of n items, $n \ll |U|$
- Hash table of size m :
 - Space $O(n \log |U|)$
 - Query time $O\left(\frac{n}{m}\right)$
- Bit array of size $|U|$
 - Space = $|U|$
 - Query time $O(1)$

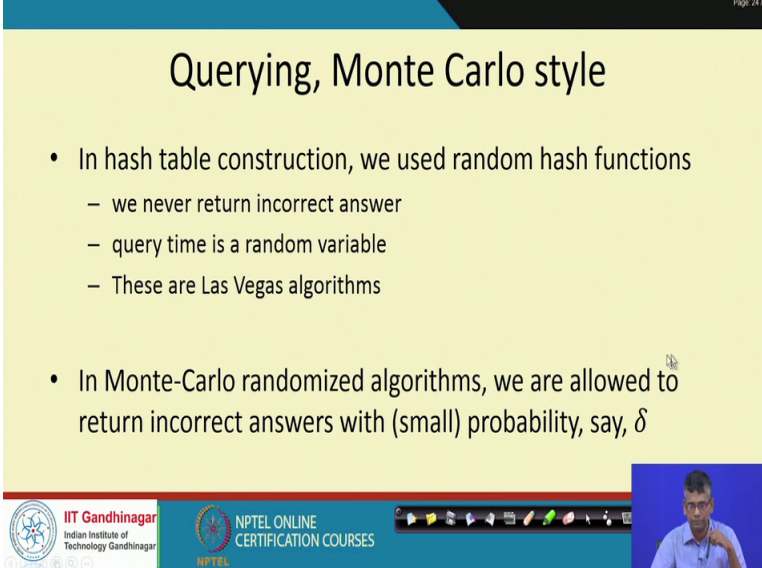
The slide footer includes the IIT Gandhinagar logo, the text "Indian Institute of Technology Gandhinagar", the NPTEL ONLINE CERTIFICATION COURSES logo, and the NPTEL logo. A navigation bar with various icons is also present.

So, let us look at some simple solutions first. So, the solutions that we looked at last, in last class was that of building a hash table. So, just to pin down the notation, we have a universe size we have universe sized U , the cardinality of U spherical. But now we are only looking to store a set of elements n , that is much less than the cardinality of U right. And we have seen that what if we created hash table. So, hash table was of size m and this m was chosen on based on the amount of memory we have accessed to. But regardless of m the space because we use chaining or something similar to that this space that, you need is really n times \log of the cardinality of U right because each of the each of the elements that you get is when you have an id and you have to store the id and the and the size of the id is \log of cardinality of U and you are going to store n of them therefore, the total space that you will use is least n times cardinality \log of cardinality of U .

Your query time however, is going to depend on your hash table right and if you happen to use at least a nice hash function then your query, time can be order n by m you cannot hope to make it much smaller than this right because there will be because its I mean this is the average \log factor, and most of the and in expectation every entry of the of your array m has this much load ok. The other extreme is to create a bit array of size cardinality of u . So, here you do not have you do not so, the ids, you know U in advance and you keep a better a of the same size of that and what do you do? You devote one bit for every element of the universe. So, the space needed is of course, just exactly the cardinality of U ; however, now you have reduced the query time to be order 1 right.

So, there is a tradeoff; there is a tradeoff that in creating hash table the space is probably quite a bit less, but a query time is little bit more in storing a bit array your space is potentially a lot, but a query time is only order 1 the question is, can we do better than either of them?

(Refer Slide Time: 03:59)



The slide is titled "Querying, Monte Carlo style" and contains two main bullet points. The first bullet point discusses Las Vegas algorithms, which are used in hash table construction. It lists three characteristics: they never return an incorrect answer, their query time is a random variable, and they are Las Vegas algorithms. The second bullet point discusses Monte-Carlo randomized algorithms, which are allowed to return incorrect answers with a small probability, denoted as δ . The slide also features logos for IIT Gandhinagar and NPTEL Online Certification Courses, along with a small video inset of a speaker in the bottom right corner.

Page 24/24

Querying, Monte Carlo style

- In hash table construction, we used random hash functions
 - we never return incorrect answer
 - query time is a random variable
 - These are Las Vegas algorithms
- In Monte-Carlo randomized algorithms, we are allowed to return incorrect answers with (small) probability, say, δ

So, it turns out that it depends right you cannot do better if you are not willing to commit any errors right. So, let us think about what the what kind of randomizations and what we did in the hash table construction. In the hash table construction we did use random hash functions; however, these random hash functions or the algorithms that we created using them never return an incorrect answer right because at the end of it we always say yes or no depending on the comparison of the actual element ids ok.

So, the query time is a random variable right because the query time depends on how sort of uniformly the hash function that you have chosen sprays the elements that are there in the in the array right; however, it does not return incorrect answer ever, and these kinds of randomized algorithms are known as Las Vegas algorithms. The name since historically I have no clear identify (Refer Time: 05:09) answer by, but that is what this called. That is when you do not return an incorrect answer, but the computational complexity of your algorithm depends on the randomization. The other style of randomized algorithms and which is that we look at now known as a Monte Carlo randomization algorithm.

So, in the Monte Carlo randomization algorithms we are allowed to return incorrect answers with a small probability; and this probability let us call it delta a something that the user will typically define to you right. So, the user will give you small probability small target error probability delta and you are allowed to return incorrect answers for this probability delta, and you have to design the choose the parameters of your algorithm so as to hit this target ok.

(Refer Slide Time: 06:07)

The slide is titled "Bloom filter" with the subtitle "[Bloom, 1970]". It contains the following text:

- A bit-array $B, |B| = m$
- k hash functions, h_1, h_2, \dots, h_k , each $h_i \in U \rightarrow [m]$

There is a handwritten note next to the second bullet point: $\{0, \dots, m-1\}$. Below the text is a diagram of a bit-array B represented as a horizontal row of 12 cells. An element x is shown above the array. Three arrows labeled h_1 , h_2 , and h_3 point from x to different positions in the array: h_1 points to the 2nd cell, h_2 points to the 4th cell, and h_3 points to the 10th cell.

The slide footer includes logos for IIT Gandhinagar and NPTEL ONLINE CERTIFICATION COURSES, along with a small video inset of a speaker.

So, here is what we will do. So, the data structure that we will look at to answer the query question is known as a bloom filter, and this was developed by Burton bloom in 1970. So, bloom filter again is a very simple data structure, there is a bit array it is basically a bit array, it is a bit array of size m ok, but now instead of one hash function there are 3 hash functions h_1 to h_k .

I am not telling you what k is right now we will get to that little later. But each h_i is a math from every element of the universe to again indices of the hash function, again indices of the array 0 to n minus 1 that is mathematically speaking each h_i is a math from U to 0 to m minus 1 which is what this expression denotes. So, we will we might use that later also ok. So, intuitively what we are going to do? When we when an element x arise, we are going to calculate now h_1 of x h_2 of x and h_3 of x in this case k equal to 3 and now we are going to look at this bits right B of h_1 of x , B of h_2 of x and B of h_3 of x and do something like that. So, what you do with it let us see.

(Refer Slide Time: 07:35)

Page 25/25

Operations

- *Initialize*(B)
 - for $i \in \{1, \dots, m\}$, $B[i] = 0$
- *Insert* (B, x)
 - for $i \in \{1, \dots, k\}$, $B[h_i(x)] = 1$
- *Lookup* (B, x)
 - If $\bigwedge_{i \in \{1, \dots, k\}} B[h_i(x)]$, return PRESENT, else ABSENT

IIT Gandhinagar
Indian Institute of
Technology Gandhinagar

NPTEL ONLINE
CERTIFICATION COURSES

So, what we do is when trying to insert an element x , we just turn on all these bits from 0 to 1 ok. That is simple this is one we are inserting, but what we do in the query right. That when a query q comes right and I calculate the $h_1(q)$ $h_2(q)$ $h_3(q)$ and what we do? I return yes if only one of them say yes do I return yes, if majority of them say yes do I return yes only if all of them say yes. So, it is not clear that is not (Refer Time: 08:14) and next try kneel the down ok.

So, here is basically the entire the entire set of operations we can do with the with the bloom filter. So, we start by initializing B very simply by setting B of i equal to 0 for all the for all the positions. When a when an element x comes, when you are trying to insert when you are trying to insert x in to into the into the bloom filter B , we calculate h_i of x for i equal 1 to k , and then we go it all this positions and set all of them to be a one that is it ok. The interesting thing happen as we saw when we are looking up, that given a particular x .

Now, x in our query which may or may not have been inserted in the in the bloom filter. Now, we are trying to say that now we are trying to answer the question that is x in the set already has x been added to the bloom filter yet right and in order to do this here is what you do? You first calculate the all the bits look at all the bits B of h_i of x . So, now, we have k bits, and we take the AND of these bits.

So, we do not look at any one of them, we do not look at the majority of them we look at their AND of this bits. And with and of this bits return present if the AND of the bit this bits is true, then you get return present and if the and say is false then you return absent clear so. So, now, let us see let us try to do some analysis of this.

(Refer Slide Time: 10:01)

Page 20/20

Bloom Filter

- If the element x has been added to the Bloom filter, then $Lookup(B, x)$ always return PRESENT
- If x has not been added to the filter before?
 - $Lookup$ sometimes still return PRESENT

h_1 x y $query$

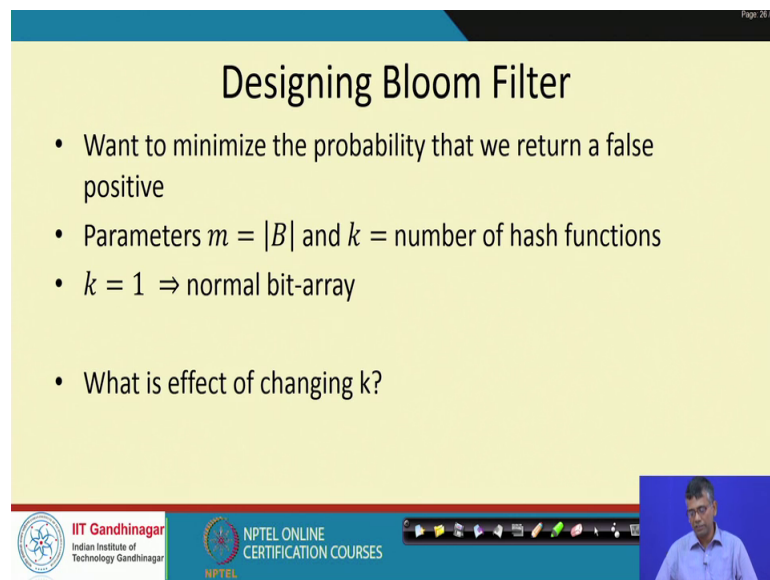
IIT Gandhinagar Indian Institute of Technology Gandhinagar
 NPTEL ONLINE CERTIFICATION COURSES
 NPTEL

So, here is the simple you know statement. If the element x has been added to the bloom filter. Then look up of $B x$ will always return present and this is very clear right. Because if the element x has been added to the bloom filter, then when we are adding it we have turned on this bits already and no other addition no other insertion turns off this bits because you have never turn off bits.

So, therefore, this bits once turned on will always remain on and therefore, look up of $B x$ will return present. So, this is very clear right now the interesting thing is the other question. What happens if x has not been added to the filter before? Is it still possible that look up of $B x$ will return present? Yes, it is unfortunately and here is a simple example, even probability constructs simpler examples though, that suppose we have 2 elements x and y right. So, think of the blue arrow as the function h_1 , think of the red as a function h_2 , think of the think of the green as function h_3 . So, so x turned on these 3 bits, y turned on these 3 bits and now a query comes. Now it is not very hard to see that if the query is being is being not using h_1 to this position if the query is being mapped using h_2 to this position, and using h_3 to this position right in that case look up of this query

will return yes because of because x and y have turned on this x . So, this is unfortunate though, but this is what we said that we can make an error the only question now is can we bound the probability of this error. Can we say that this the probability of this happening is really small.

(Refer Slide Time: 12:07)



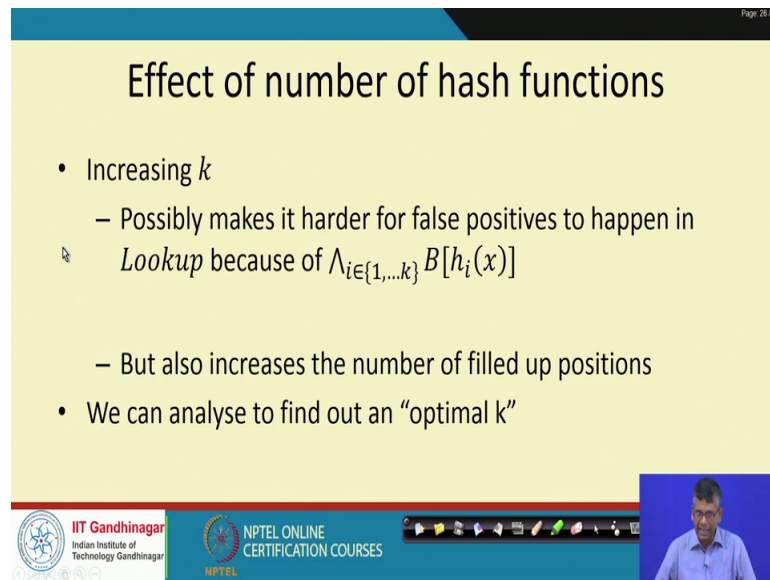
The slide is titled "Designing Bloom Filter" and contains the following bullet points:

- Want to minimize the probability that we return a false positive
- Parameters $m = |B|$ and $k =$ number of hash functions
- $k = 1 \Rightarrow$ normal bit-array
- What is effect of changing k ?

The slide footer includes the IIT Gandhinagar logo, the text "IIT Gandhinagar Indian Institute of Technology Gandhinagar", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset shows a man in a blue shirt.

So, and this is what we call a false positive right that we return the positive answer, but that is false. So, what are the parameters of bloom filter? The parameters are the 2 parameters that are really at hand are the size of the bloom filter m and this mysterious number k , which is the number of hash functions. So, just to sort of this (Refer Time: 12:34) check recall that if I were just using k equal to one that is really it is a normal bit array right that is nothing interesting there going on. So, what is the effect of changing this k ? k cannot really be decreased to be less than 1, but what will happens when k is increased? So, if you think about it really 2 things are happening.

(Refer Slide Time: 12:56)



The slide is titled "Effect of number of hash functions". It contains the following text:

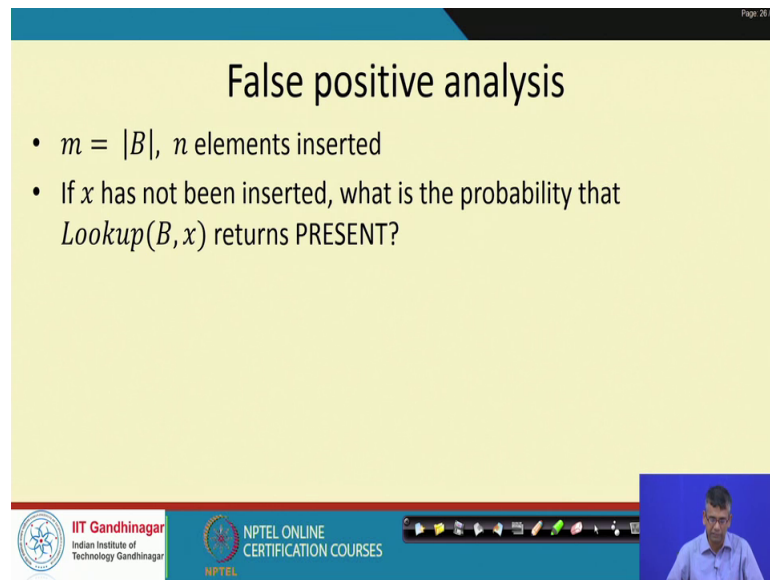
- Increasing k
 - Possibly makes it harder for false positives to happen in *Lookup* because of $\bigwedge_{i \in \{1, \dots, k\}} B[h_i(x)]$
 - But also increases the number of filled up positions
- We can analyse to find out an “optimal k ”

The slide footer includes the IIT Gandhinagar logo, the text "IIT Gandhinagar Indian Institute of Technology Gandhinagar", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset in the bottom right corner shows a man in a blue shirt speaking.

Increasing k , possibly makes it harder for false positives to happen right because look up the when you call the look up functions, it looks at k bits and takes the AND. So, if k is large then the chance that any one of this bits has been turned on by some other element is going to decrease as k increases. So, therefore, that is a good thing right therefore, increase in k potentially decreases the chance of false positives to happen or rather the probability of false positives, it decreases the probability of the false positives; however, it is not a straight forward unfortunately.

Because if you think about it, if you increase k then you are filling up more and more of the hash table more and more of this bit array right. So, in extreme case if k equal to n is even one element turns on all the bits right. So, therefore, increasing k is not always good; and then the next question is that is there an optimal way to choose this k right? Then is there a switch spot such that if I choose the k according to this particular value right that sort of balances have these 2 effects, that sort of gives me some effects of sort of decreasing the false positives in the look up as well as it does not, it may be does not fill up all the increase. So, fast so. So, it is a way to choose k in a smart way right.

(Refer Slide Time: 14:32)



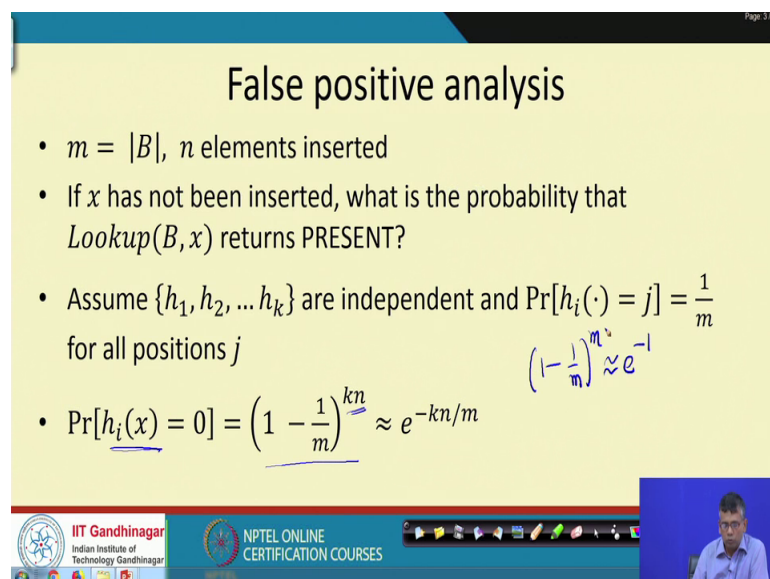
Slide 26/28: False positive analysis

- $m = |B|$, n elements inserted
- If x has not been inserted, what is the probability that $Lookup(B, x)$ returns PRESENT?

The slide includes logos for IIT Gandhinagar and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of the presenter.

So, in order to analyze this, we need to do we need to analyze the event of what is happening, when we call look up of B x right. So, if x has been inserted this nothing much to do because we know that look up of the x will always return present; however, let us look at what happens if x is not been inserted before, but now we are querying it and so, we want to burn the probability that look up of B x returns present, because that is the bad event that is false positive event ok.

(Refer Slide Time: 15:09)



Slide 27/28: False positive analysis

- $m = |B|$, n elements inserted
- If x has not been inserted, what is the probability that $Lookup(B, x)$ returns PRESENT?
- Assume $\{h_1, h_2, \dots, h_k\}$ are independent and $\Pr[h_i(\cdot) = j] = \frac{1}{m}$ for all positions j
- $\Pr[h_i(x) = 0] = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$

Handwritten note: $\left(1 - \frac{1}{m}\right)^m \approx e^{-1}$

The slide includes logos for IIT Gandhinagar and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of the presenter.

So, I mean for this analysis, let us start out the assuming that h_1, h_2, \dots, h_k are all fully independent are all independent of each other we will talk about what happens if they are not completely independent.

So, and let us also start out the assuming, that probability that for any that for any event that for any of this h is the probability that h of i that the probability that h_i of x equals j for a fixed j and for a fixed x equals $1/m$ for all x and j . That is all that really says is that even any x , the probability that it maps to a particular i its equally likely to following its equally likely to map into any of the positions of the array that is all this says. So, under this assumption, when we have inserted n elements right suppose, if we have inserted n elements. So, we have turned on k time's n bits right because each element insertion turns on k bits.

So, now, and this k bits so, for choosing each of these bit has been chosen using uniformly among the bits from 1 from 0 to n minus 1 . So, now, let us look at the n the bit the bit h_i the position h_i of x ; now let us look at the position h_i of x ok. So, what is the probability thus that this bit is still 0 ? So, if this bit is still 0 , the only way it could happen is that all the $k n$ insertions that are happened in until now have happened in the other n minus 1 positions right? That that none of the kn insertions until now have try to turn this bit to be on and the probability of that event is $1 - 1/m$ to the power kn right and this approximately is $e^{-kn/m}$ and we will use this approximation, we get this because $1 - 1/m$ to the power m .

You can write it as approximately $e^{-kn/m}$ and therefore, $1 - 1/m$ to the power kn is like $e^{-kn/m}$ right. So, this is a very tight approximation you can do slightly better analysis than this, but this is what we are stick to.

(Refer Slide Time: 17:47)

The slide is titled "False positive analysis" and contains the following content:

- The expected number of zero bits $\approx me^{-kn/m}$ w.h.p.
- $\Pr[\text{Lookup}(B, x) = \text{PRESENT}] = (1 - e^{-kn/m})^k$
- Can we choose k to minimize this probability

Handwritten annotations on the slide include $h_1(x)$, $h_2(x)$, and $h_k(x)$ above the probability equation, and a blue circle around the term $(1 - e^{-kn/m})^k$. The slide footer includes logos for IIT Gandhinagar and NPTEL ONLINE CERTIFICATION COURSES, along with a small video inset of a speaker.

So, now we are saying that the probability of a single bit being 0 is e to the power minus kn by m right. So, now, look up when we look up return present, look up will return present if all the k bits if all the k bits h_1 of x , h_2 of x and until h_k of x all of them say 1 right. So, the probability of that is that all of each of them are individual 1 and the probability that any one of them is 1 is $1 - e$ to the power minus kn by m , because the probability that it was 0 was e to the power minus kn by m . So, therefore, the probability that h_1 of x the bit h_1 of x is one is $1 - e$ to the power minus kn by m .

So, therefore, the probability is that all these k bits each all of them are 1 is nothing, but $(1 - e$ to the power minus kn by m) to the power k ok. So, notice this is not entirely accurate, because it is possible that h_1 of x equals h_2 of x ; however, these are all marginal cases and, but analyzing this marginal cases needs a little more heavy machinery that we would not go in to for now let us just stick to this almost correct analysis so. So, now, we have the probability of a false positives to be the probability that; to be the probability that the look up of Bx is present to be this expression $(1 - e$ to the power minus kn by m) to the power k right. So, notice that this expression very nicely captures the 2 roles of k right because as k changes, the inner expression the inner expression $1 - e$ to the power minus kn by m increases as k changes one way, monotonically as k changes whereas, this to the power k definitely decreases.

So, the expression inside actually increases, but the expression outside, but the, but the expression outside tends to pull k tends to pull the value of this of these particular the power of k tends to pull the value down ok. So, the question is that is there a switch spot can you choose k to minimize this probability ok.

(Refer Slide Time: 20:23)

The slide is titled "Choosing number of hash functions". It contains the following content:

- $p = e^{-kn/m}$
- Log (False Positive) =

$$\log(1 - p)^k = k \log(1 - p) = -\frac{m}{n} \log(p) \log(1 - p)$$

Minimized at $p = \frac{1}{2}$, i.e. $k = m \log(2)/n$

A plot of a parabola is shown on the right side of the slide, with the x-axis ranging from 0 to 1.0 and the y-axis ranging from -0.5 to -0.1. The parabola opens upwards and has its vertex at $p = 0.5$. The plot is labeled "Computed by WolframAlpha".

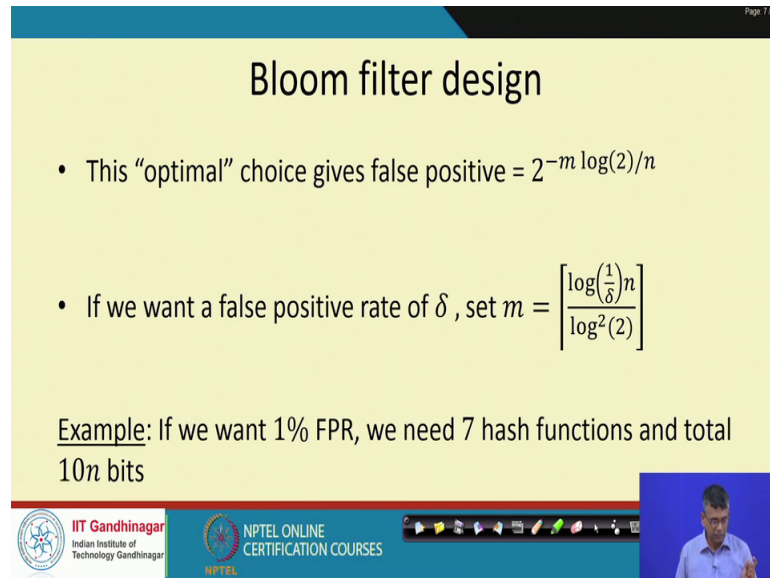
The slide footer includes the logos of IIT Gandhinagar and NPTEL ONLINE CERTIFICATION COURSES.

So, now it is actually very simple, let us have a short hand notation let us write p to the p equal to e to the power minus kn by m and minimizing a probability is the same as minimizing the log of that. So, let us take the log of the false positives log of the false positives is nothing, but this log of 1 minus p to the power k because p verses expression and k come. So, it becomes k log 1 minus p ok. So, and if you plug in the value of; if you plug in the value of k using the definition of p you get minus m by n log p log 1 minus p ok. So, now, it is really simple question of calculus right, but we have going even lazy what we will say? Let us just plot this expression log p times log of 1 minus p minus log p tends log m of 1 that is p.

And there is a plot that I have generated in using from alpha is very easy to see that this expression is minimized at the midpoint that is at p equal to half ok. You should really take the derivative of this and (Refer Time: 21:32) 0 1 and then and check the check the second derivatives and so on, but am just being lazy. So, this expression is minimized at p equal to half, which means at the optimal value of choosing p and therefore, that of

choosing k is really p equal to half which means that k is $m \log 2$ by n . So, here we are using the natural base for \log ok.

(Refer Slide Time: 22:00)



The slide is titled "Bloom filter design" and contains the following content:

- This "optimal" choice gives false positive = $2^{-m \log(2)/n}$
- If we want a false positive rate of δ , set $m = \left\lceil \frac{\log\left(\frac{1}{\delta}\right)n}{\log^2(2)} \right\rceil$

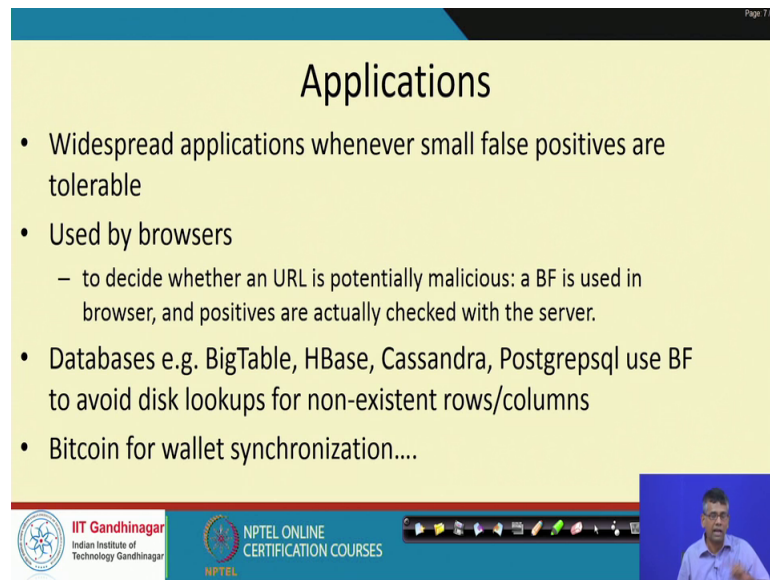
Example: If we want 1% FPR, we need 7 hash functions and total $10n$ bits

The slide footer includes logos for IIT Gandhinagar, NPTEL ONLINE CERTIFICATION COURSES, and NPTEL, along with a small video inset of a speaker.

So, this optimal choice of k gives you a false positives that is 2 to the minus $m \log 2$ by n and this you just get by plugging in the value ok. So, now, if you want a false positive of δ , but supposing you all are get an engineer and your manger is come to you and sort of and ask you to design bloom filter for false positive rate of δ right. Now you know what to do you get to use k and you have to choose m right. So, m and you tell him that I am going to need this much I am going to need machine that that can hold this size of bit array. In the size of m that you need is \log one by δ time's n divided by \log square 2.

So, all of these you just get by plugging in the value of k and by plugging in the value of m . So, for example, if you want one percent false positive rate, which is potentially very small you only need seven hash functions and the total of $10 n$ bits right. So, now, what we have is a data structure that has only constant times n number of bits right remember n is not the size of the universe, n is the number of elements that we have understood right. And you need something like a constant times that for a small enough of constant and you need 7 hash functions. So, at query time you only need to calculate 7 hash functions of the query and you look these up. So, it is fairly fast.

(Refer Slide Time: 23:28)



The slide is titled "Applications" and lists several use cases for Bloom filters. It includes logos for IIT Gandhinagar and NPTEL Online Certification Courses. A small video inset shows a speaker in the bottom right corner.

- Widespread applications whenever small false positives are tolerable
- Used by browsers
 - to decide whether an URL is potentially malicious: a BF is used in browser, and positives are actually checked with the server.
- Databases e.g. BigTable, HBase, Cassandra, Postgresql use BF to avoid disk lookups for non-existent rows/columns
- Bitcoin for wallet synchronization....

So, bloom filters have a have widespread applications whenever false positive rates are tolerable. In fact, a very celebrated use is by was by this web browser chrome. So, what it did do? It is it used to should have bloom filter to against for potentially malicious URLs ok. So, this bloom filters.

So, suppose you type a URL in the in the browser I mean in the chrome browser right. So, now, the chrome browser is going to give you a warning with typically still does to say that, this URL is potentially malicious ok. So, how does it do that? What it does is that it creates a bloom filter of the malicious URLs that it has and it shifts to each of the clients it shifts to each here by (Refer Time: 24:14) has one of this potentially has one of this filters. So, when you type this query, when you type this URL; this URL is checked against this clients and bloom filter where the clients.

And bloom filters says no that this is not malicious then you can just go ahead, but if the client side becomes bloom filters do not have false negatives. But if the clients side bloom filters says yes this is potentially malicious well it is at that point chrome sends this URL to the server and does an actual check because and because the probability of false positive is small right the number of times it is contacting the server, it decreases right. It decreases I mean beyond this naive solution, if you are contacting the server with every with every potential URL right.

So, similarly I mean it is used a lot in all these distributed databases such as BigTable, HBase, Cassandra. What they have is that they have their data columns that are distributed over multiple machines ok. And now Cassandra for instance given a particular given a particular column right. So, suppose you are asking that does this machine contain the column right.

So, rather than doing anything else Cassandra first has a bloom filter that it carries around and it tries to answer yes no question using this bloom filter. Now there is bloom filter says no, then that is it where the bloom filter says yes at that point Cassandra does some more competitions in order to sort of 2 in order to decide whether to actual do a disk look of for that particular column in this machine. It is also used in bit (Refer Time: 25:53) valets synchronization between one point of the details of that so.

(Refer Slide Time: 26:02)

The slide is titled "Handling deletions" and is page 7 of 32. It contains the following text:

- Chief drawback is that BF does not allow deletions
- Counting Bloom Filter [Fan et al 00]
 - Every entry in BF is a small counter rather than a single bit
 - $Insert(x)$ increments all counters for $\{h_i(x)\}$ by 1
 - $Delete(x)$ decrements all $\{h_i(x)\}$ by 1
 - maintains 4 bits per counter
 - False negatives can happen, but only with low probability

A diagram on the right shows a horizontal bar representing a counter, with two arrows pointing to it from above. Below the bar are two vertical bars representing bit counters, with arrows pointing from the horizontal bar to each of them.

The slide footer includes the IIT Gandhinagar logo, the text "IIT Gandhinagar Indian Institute of Technology Gandhinagar", the NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a man speaking.

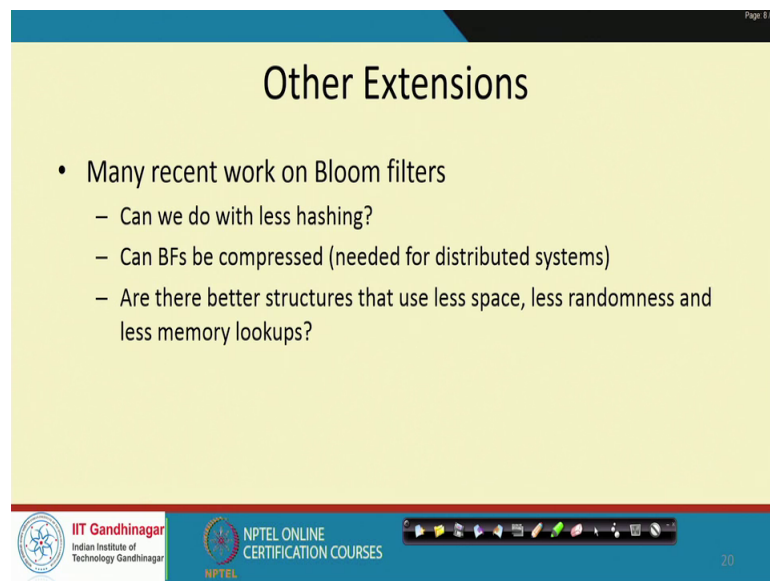
So, all very nice, but there is one particular problem that you might already have noticed what do we do, if you want to delete an element right? And I never showed you what to do for deletion improve filters and it is not surprising that in bloom filters you cannot delete, in ordinary bloom filters you cannot delete because once you delete right your analysis of our analysis of the optimal k goes away.

Because now, although I mean you might have false negatives also, because although you have inserted an element may be you deleted one of the bits you made one of the bits to be to be 0 when you delete a some other element right. So, therefore, this is not

obvious a solution has given in terms of this counting bloom filter, and it is very intuitive or it says is that ok. Previously each element of the bloom filter is to contain a single bit let us now add to it a small counter. So, think of it as follows that we have the we have the array, we have the array and now we have a small counter of let us say 4 bits with each position of the array ok. So, now, when we add an element right we will go to the counter and we increment that particular counter right.

So, will. So, for every i from 1 to k , we will go to the corresponding counter and incremented by 1. Now when we going to delete an element again we will go to that counter go to this corresponding counters and decrement this counters by 1 and this counters will be small, this counters will be will be like 4 bits per counter ok. So, it is possible to show using analysis that is almost similar to that of bloom filters little more complicated that, that this is actually fine using 4 digital counters is fine in most case false negatives can happen, but with very small probability and false positives analysis of false positives is similar to that our bloom filters.

(Refer Slide Time: 28:16)



Page 8/13

Other Extensions

- Many recent work on Bloom filters
 - Can we do with less hashing?
 - Can BFs be compressed (needed for distributed systems)
 - Are there better structures that use less space, less randomness and less memory lookups?

IIT Gandhinagar
Indian Institute of
Technology Gandhinagar

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

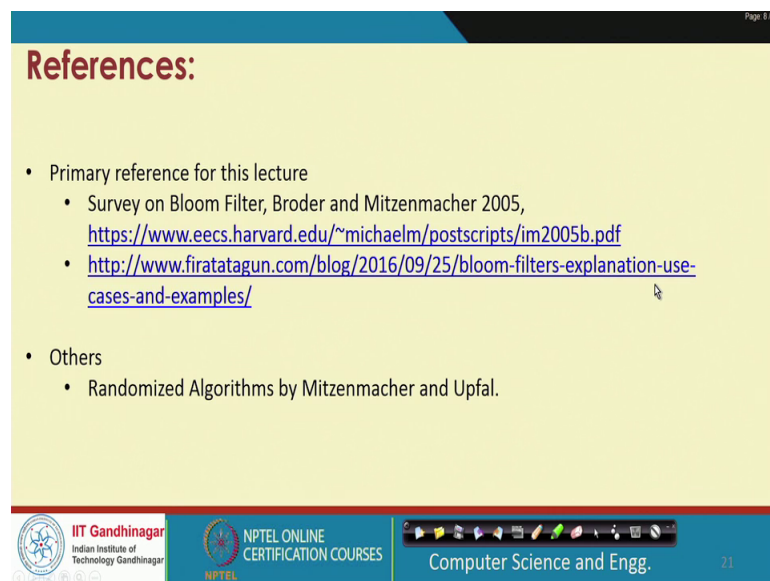
20

So, and there are a lot of other very interesting extensions for bloom filters. Here is the very interesting question that, we used for every input element that comes we hashed it k times using k hash functions right can we do with hashing much less hashing may be on with 2 hashes? Since possible actually a very interesting (Refer Time: 28:38) case of bloom filters is when you are as you mention these when it is used in the distributed

systems. In that case sometimes what happens is that, you want to compress this bloom filters and sent it across right.

Because in the in case of Cassandra, if Cassandra was to storing that that this machine has these columns of the data and in case of bloom filter using that, it might found a store that machine might find send that bloom filter to other machines after compression, it answer that bloom filters can be compressed very nicely. Furthermore there are other structures the theoretically at least, use less space less randomness and sometimes less memory look ups. However, most of the structures are really not very practical yet, they have been I mean they kind of work for ranges of error probabilities and the data size that are I mean for all practical purposes bloom filters, that the performance of bloom filters are still bits the performance this more theoretical structures. However, if you are interested in research this is away, this is an excellent area to look into and.

(Refer Slide Time: 29:48)



Page 8/12

References:

- Primary reference for this lecture
 - Survey on Bloom Filter, Broder and Mitzenmacher 2005, <https://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf>
 - <http://www.firatatagun.com/blog/2016/09/25/bloom-filters-explanation-use-cases-and-examples/>
- Others
 - Randomized Algorithms by Mitzenmacher and Upfal.

IIT Gandhinagar
Indian Institute of
Technology Gandhinagar

NPTEL ONLINE
CERTIFICATION COURSES

Computer Science and Engg. 23

The primary reference for this lecture is the survey of bloom filter by the excellent survey by Andrei Broder and Mitzenmacher and the some of the used cases and pictures that we are drawn from this particular block site. There also very nice analysis of bloom filters in the in the book Randomized algorithms. So, that is it for today.

Thank you.