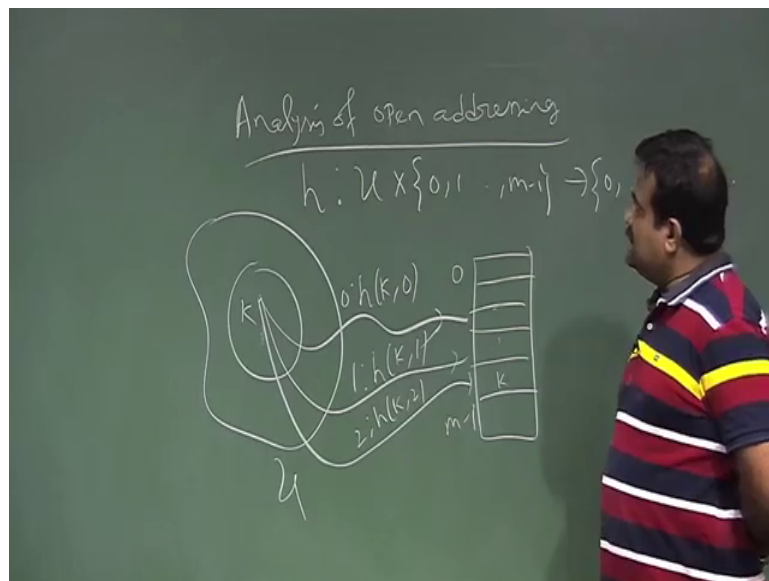


An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 23
Universal Hashing

So we talked about universal hashing. So, before that let us just complete the analysis of open addressing, so analysis of open addressing.

(Refer Slide Time: 00:28)

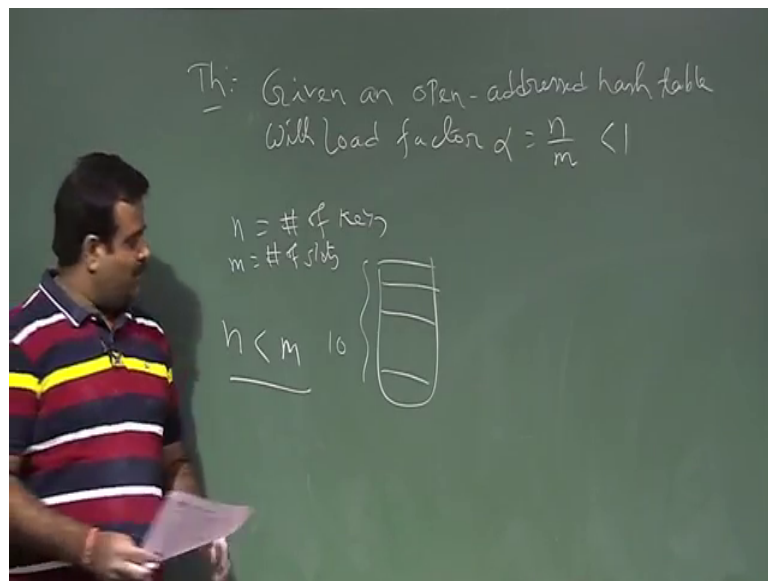


So, basically just to recap, so in the open addressing method we are just having the probe sequence here hash function is a function from $U \times \{0, 1, \dots, m-1\}$ to $\{0, 1, \dots, m-1\}$. So, we have basically, we have a hash table of size m 0 to m minus 1 . So, I mean this is the set of keys now if you insert a key. So, what we do we just apply the 0 th probe. So, so we apply the 0 th probe. So, h of k comma 0 and if it is heating some occupied slot then we go for the next probe h of k comma 1 this is the 0 th probe, this is 1 th probe and if it is still heating some occupiers slot then we go for the next probe h of k comma 2 and these way we continue until we get a empty slot. Also you get a empty slot we will insert that value k inside the k over here.

So, and we have seen to example of probing linear probing and the double probing in the last class. So, now, we will analyse is this is called open addressing. So, we analyse the, so everything we have to be feed in the table. So, there is no storage outside the table.

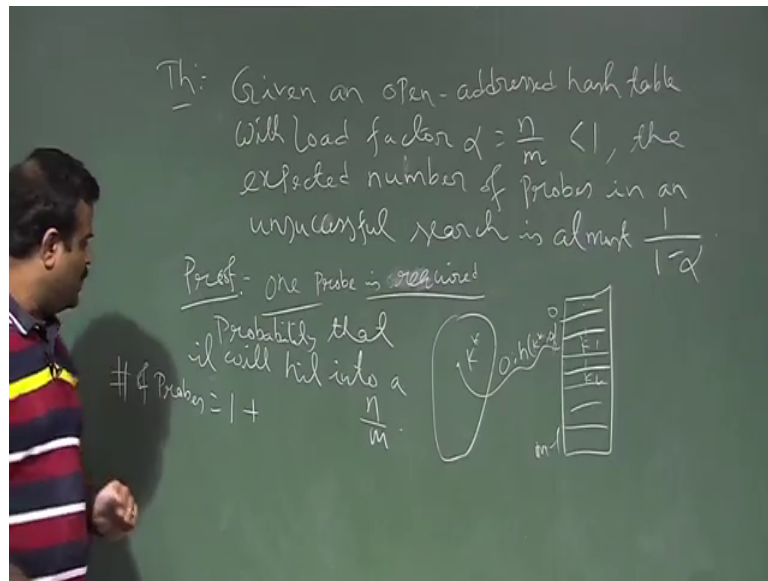
So, we are not allowed to have a linked list outside the table. So, changing is not allowed. So, we have to feed everything inside the table. So, basically the idea is to search for a empty slot by this probe sequence. So, we have a sequence of hash functions. So, we have applying first hash function if it is heating some occupied slot next expansion like this. So, this is the recap of what is open addressing now let us analysis this open addressing by a theorem.

(Refer Slide Time: 03:00)



So, this theorem is telling, given an open address hash table hash table with load factor alpha which is basically n by m and this has to be less than 1. Why it is less than 1? Because if there are, so if there are n keys, if there are m slots if the keys number of keys is more than the slot then there is no way we can feed the keys, if there are say if there are ten slots and if there are 20 keys then we cannot fit 20 keys in the 10 slot. So, we need to have the extra storage there so that assumption is mandatory. So, n is the number of keys and m is the number slots. So, n has to be less than m otherwise we cannot fit the keys in the table by the probing because if it is so, this assumption is quite.

(Refer Slide Time: 04:35)



So, n has to be less than m then the theorem is telling the expected number of probe n for around successful search the expected number of probes in an unsuccessful search, unsuccessful search means we are searching a key which is not there in the table. So, this is theorem what this theorem is telling. So, we have n keys we have m slots and we have a open address hash function and we have fill up these keys into the slots by using this open address hash function and now suppose we are trying to find out the key which is not there in the table. So, now what is the number probes we should have we should do this for this search. So, that is the theorem.

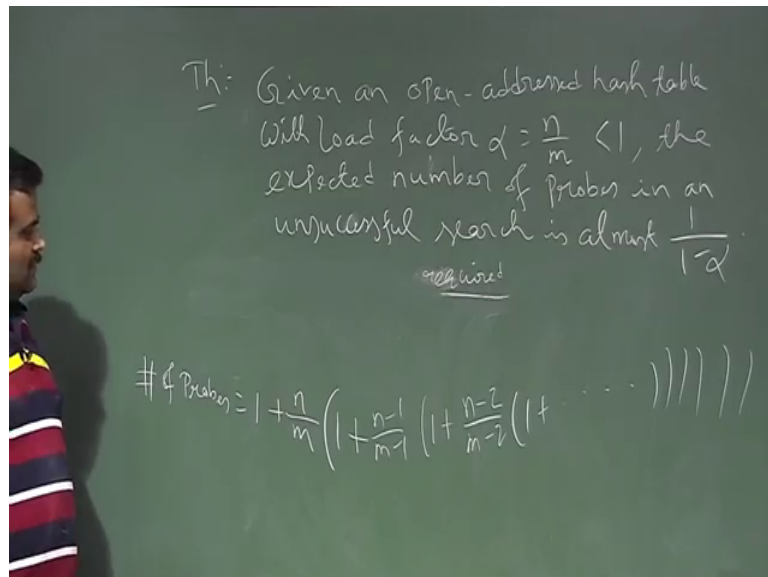
And that it is telling 1 by 1 minus α . So, how to proof this? To proof this we will just use some probability staff. So, there are say basically there are m slots 0 to m minus 1 m slots and in this m slots there are k keys I saw n keys are seating there now 1 probe is mandatory. So, one probe, probe is required, required. So, one probes is mandatory because. So, suppose we are finding a key k a star which is not there in the table. So, for that we need to try the probe that is 0 of, 0 that is the 0 th probe. So, 1 probe is required. So, if we denote this is the number of probes. So, one probe is required.

Now, when we go for the second probe if this is heating to a occupied slots if this is heating to a occupied slots then only we go for the second probe I mean next probe. So, now, what is the probability that it is heating to a occupied slots. Now what is the

probability that it will heat to a occupied slot now total number of slot is same. Now there are n keys live in this here, so just the classical definition of the probability. So, if we choose one of this if it is heat one of the slot where this n keys are lives then that is the probability. So, basically probability that it will heat to a occupies slot is n by m.

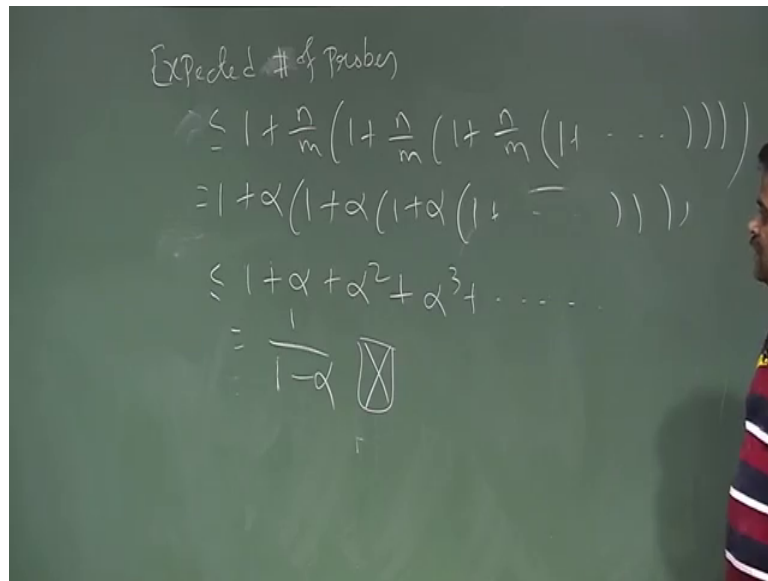
So, the probability that probability that it will it will heat to a into a occupied slot is basically n by m because there are a m slot which is favourable case and there are m slots are there n keys are there in that slot and there are total m slots. So, if we can heat anyone of this n. So, that is the just classical definition, so n by m. So, with this probability n by m then we will go for the next probe so that means, we will go for next probe with this probability 1 by m and then again next probe.

(Refer Slide Time: 09:14)



Then again we will go for the next probe if it is again heating to a occupied slot and that probability is basically n minus 1 by m minus 1 and then we will go for the next probe like this. And again we will go for the next probe it is n minus this probability like this. So, this is the number of probes or we can say expected number of probes in the probability calculation. So, now, we want to simplify this.

(Refer Slide Time: 10:03)



Expected # of Probes

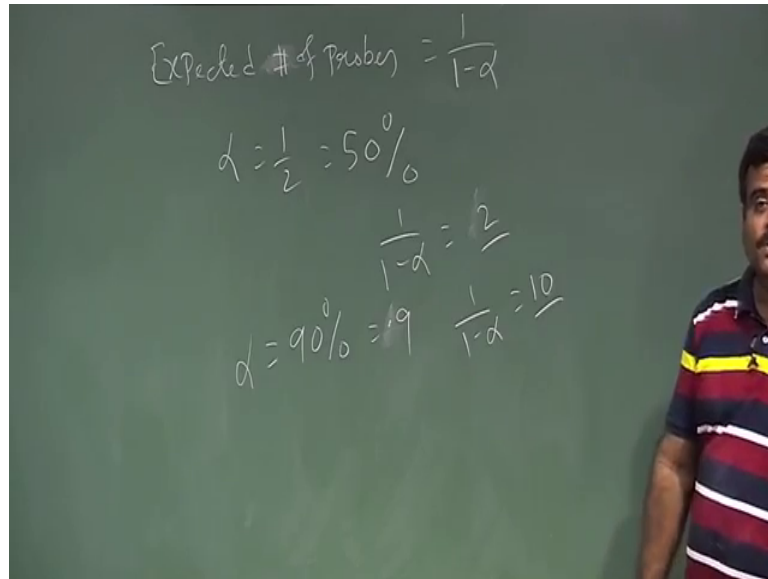
$$\leq 1 + \frac{n}{m} (1 + \frac{n}{m} (1 + \frac{n}{m} (1 + \dots)))$$
$$= 1 + \alpha (1 + \alpha (1 + \alpha (1 + \dots)))$$
$$\leq 1 + \alpha + \alpha^2 + \alpha^3 + \dots$$
$$= \frac{1}{1-\alpha}$$

So, this is the, so the expected number of probes expected number of probes for an unsuccessful search is basically this. Now we can simplify this just by using this fact n minus i by m minus i is less than equal to n by m for all i this can be easily probe.

So, if we use that. So, this is basically less than equal to $1 + 1 + \frac{n}{m} + 1 + \frac{n}{m} + \frac{n}{m} + \dots$. So, this $\frac{n}{m}$ is basically α this is $1 + \alpha + \alpha + \alpha + \dots$. So, now, we have the expression like this is less than basically 1 , so this is basically $1 + \alpha + \alpha^2 + \alpha^3 + \dots$. I mean we can say less than now this basically 1 by $1 - \alpha$ since α is less than 1 . This is the power series, this is the some infinite some, this is 1 by, so this is the probe.

So, expected number of proof for a unsuccessful search in a open address hash table is basically 1 by $1 - \alpha$. So, what is the meaning of this? So, this is the 1 by $1 - \alpha$.

(Refer Slide Time: 11:59)



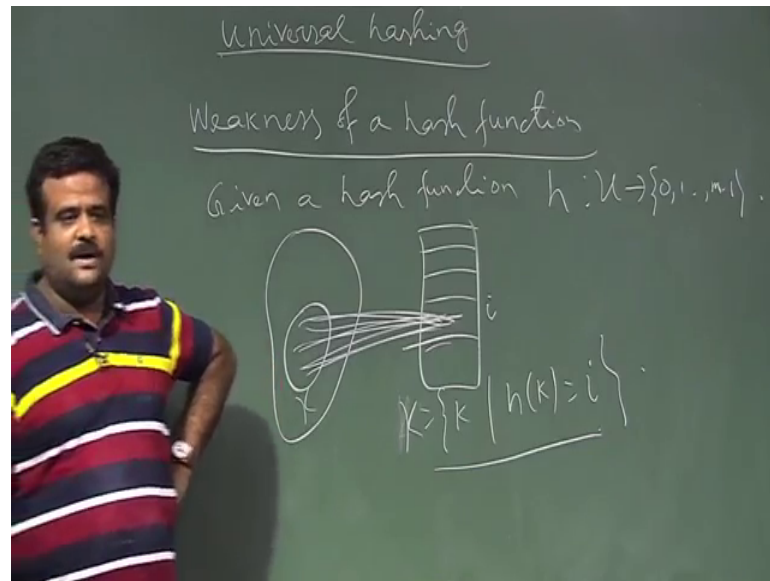
Expected # of Probes = $\frac{1}{1-\alpha}$
 $\alpha = \frac{1}{2} = 50\%$
 $\frac{1}{1-\alpha} = \frac{2}{1}$
 $\alpha = 90\% = .9$ $\frac{1}{1-\alpha} = \frac{10}{1}$

So, now suppose our table is half full. So, suppose our alpha is 50 percent so that means, our table is half full. So, if we have hundred slots suppose there are 50 keys half full. So, alpha is 1 by half 1 by 2. So, then what is this value than 1 by 1 minus alpha is basically 2 so that means, we need just two probes for an unsuccessful search, but what happen if the table is 90 percent full, I mean this is 1 by this is 0.9, so 9 by 10, if the table is 90 percent full then what is 1 by 1 minus alpha this is basically 10. So, if the table 90 percent full then we need to have 10 attempt 10s probes for an unsuccessful search.

So, this is quite obvious because if we have 90 percent full means if there are 100 slot and if there are 90 keys and which are distributed about this slots then it is almost loaded. Then for a search key which is not there in the table we need to have the 10 probes for this. So, this is quite, so this is the analysis of this open addressing and it is good in the terms of this handling the collision because collision will be there in the hash function we cannot say you can construct a hash function there will be no collision because hash function this doming size is big then the co doming size. So, there has to be collision. So, collision this is the one way we can handle the collision by using the open addressing.

So, now we will start what is called universal hashing or universal hash function.

(Refer Slide Time: 14:07)



So, to start this we will talk about a weakness of a hash function this came from this weakness of a hash function. So, suppose there is a computation you have team a team b suppose IBM came to your place and announce a competition to have a hash function to build a hash function. So, they are going to use this for their or for their often hash function is haven use in compiler operating system. So, suppose they announce a computation to have a hash function, so you develop hash function and b team develop another hash function and you both submitted the hash function to IBM.

So, now, what IBM will do IBM will give your hash function to b and your hash function to team a and ask for a, ask to find set of keys where it is colliding. So, it is always possible to have this because we know there will be the collisions.

So, if we have time we can always. So, given a hash function, given a hash function h which is basically function form, so one can always construct a portion of the key where it is collide a because if you have a time you can try for all possible keys and then you can just have this key which is basically for all values it is colliding into same slot. So, and you have to in the you have to in the computation. So, you obviously, will try for all possible key for your opponent code and you will come out with the such a key and you will submit this to the IBM that this is the set of keys were my friends code is not working that and your friend is not seating ideal. So, they are also trying to find out a set of keys where your code is performing bad. So, that is can be possible because if you

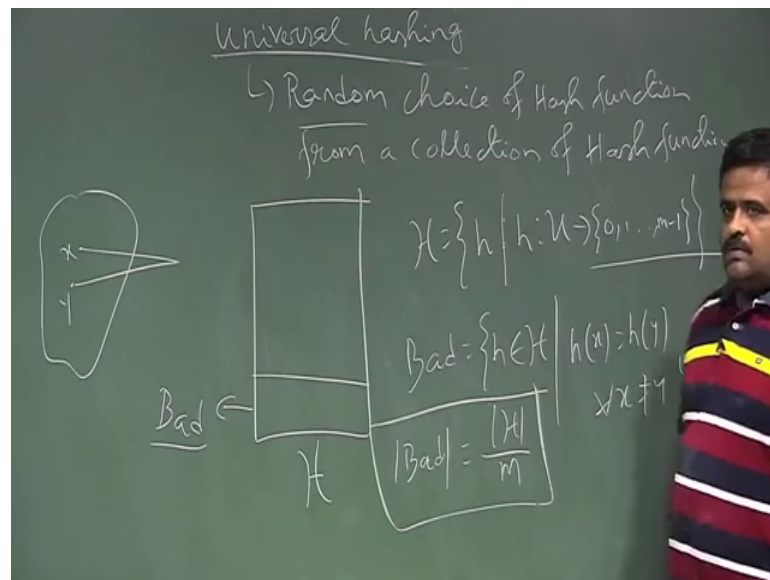
know the hash function in hand then you can try for all possible keys and then you can come with some portion where your code is performing badly.

So, this is a fundamental weakness hash function. So, to avoid this weakness, so one can always construct this, this h such that h of k is always going to be a fixed slot, so each number of collision. So, now, to avoid that how we can avoid this scenario how our friend cannot our friend or adversary cannot come with some I mean cannot come with some input where my code is performing bad. So, this type of thing we did in a quick slot if you remember in the quick slot if you know the position of the p board element the say in our original version of the partition we are choosing the first, had a first element as a p board element.

And then if we put the minimum or maximum in the first one then we are always getting the worst case. So, we can even if we know the position is the third element third means not third smallest third position index third then also one can put the minimum maximum in the third element always and can get an input can construct an input where it will give a 0 is to n minus 1. So, if we know the position of the p board element then we can have an example where our code is performing bad. So, to avoid that what we did in the quick slot we did a randomised choice of the p board element. So, we choose the p board element randomly position of the p board element randomly from the given array. So, here also we will do the same thing.

So, to avoid this we have to choose a hash function randomly from a collection of the hash function. So, this is the way we can avoid this weakness because if we choose the randomly hash function then our friend is not knowing which hash function we are going to choose at the wrong time because this choose is at the wrong time. So, nobody can come with some set for which it will perform badly because we do not know which hash function we are it is going to choose randomly. So, we have a collection of a hash functions we are going to choose hash function randomly. So, that is the idea behind this universal hashing.

(Refer Slide Time: 19:30)



So, let us talk about this little more. So, this is to overcome this weakness. So, random choice of the, random choice of hash function from a collection of hash function, so that collection has some property what is that suppose we have a collection of hash function, this is a H . So, H is basically collection of hash function it is the function such that u cube. So, we have, so we considered set of all hash function I mean the collection of hash function such that among this collection suppose there is a portion which is referred as a bad portion that in the sense now if we chose a hash function from this portion then there will be collision; that means, if h is coming from this. So, bad is basically, so if we choose two keys.

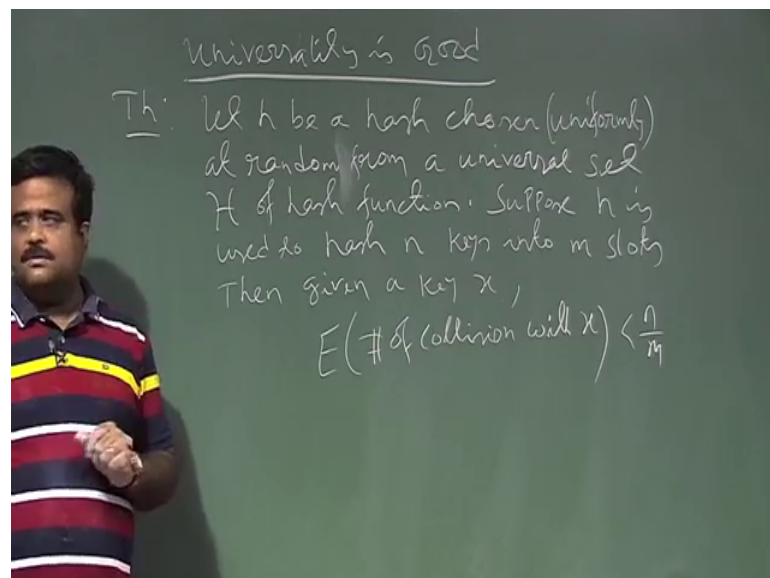
So, this is the keys set if we choose two key x y , x and y then if we choose a hash function which is x not equal to y if you choose a hash function from the form this bad portion then there has to be collision so that means, bad means it is basically subset of H such that h of x equal to h of y for all x not equal to y . So, this is called bad portion; that means, if our hash function is coming from this portion then there as to be collision for any two key any two distinct key, if we choose any two distinct key then definitely there will be colliding into the there will be the collision. So, this is called bad portion I mean I refer this as a bad portion because it is giving as a collision for sure.

Now, if the cardinality of bad portion is basically cardinality of H by m . If the cardinality of this set that mean cardinality means number of element in this set is basically 1 by m

fraction of the total then this collection is called universal collection. If the cardinality of this bad portion is 1 by m fraction of the total n is the number of slots, 1 by m fraction of the total then this collection is called universal collection of hash function and if we choose a hash function from this collection and that hash function is called universal hash function. So, now, we will talk about why this universal hash function is good. So, the bad portion is the 1 by m fraction of the total then this collection is called universal hash collection and if we choose a hash function randomly from this collection then it is called universal hash function or universal hashing.

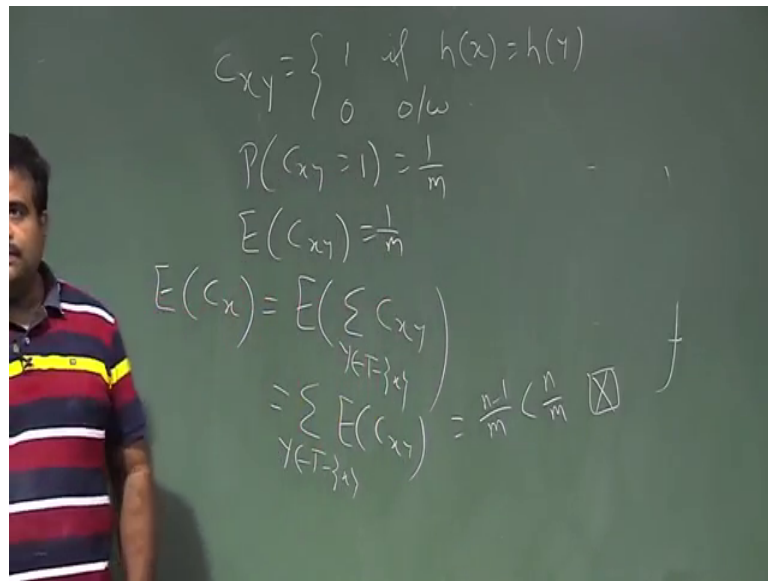
So, next we will talk about why this universality is good why you should take this hash function randomly from this collection. So, that is that is the next topic.

(Refer Slide Time: 23:20)



Next we will talk about why universality is good universality is good. So, this we will proof by a theorem again. So, this theorem is telling let h be a hash function chosen uniformly at random equally (Refer Time: 23:58) at random from a universal set h of hash function all hash function and suppose h is use to hash in arbitrary key n keys into m slots. Then given a key x , given a key x we have expected number of collision with x is less than n by m which is basically the load factor and this good because if we even we cannot expect we are expecting the number of collision is less than n by m which is the load factor and that is good.

(Refer Slide Time: 25:49)


$$C_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{o/w} \end{cases}$$
$$P(C_{xy} = 1) = \frac{1}{m}$$
$$E(C_{xy}) = \frac{1}{m}$$
$$E(C_x) = E\left(\sum_{y \in T, y \neq x} C_{xy}\right)$$
$$= \sum_{y \in T, y \neq x} E(C_{xy}) = \frac{n-1}{m} \left(\frac{n}{m}\right) \square$$

So, how to proof this? So, to proof this we need to use some indicator random variable. So, we choose the hash function from this universal set universal hash function set uniformly at random. So, we denote this random integral random able C_{xy} is basically 1 if h of x equal to h of y and 0 otherwise.

Now what is the probability of C_{xy} ? Is 1, is basically 1 by n because we are fixing x and it will colliding to this, so we are choosing the hash function from this universal set. So, this is our H and this is the bad portion and the cardinality of the bad portion is cardinality of H by n . Now if we have a x now if we choose a y which is not equal to x now what is the probability that collision. So, pro collision will happen if we choosing the hash function from this. So, size of this is H by m and size of total is H . So, the probability that there will be the collision is 1 by m basically because H by m divided by H . So, it is basically 1 by m . So, this is the probe, so expected value of c of x is basically 1 by m .

Now, we are looking for expected value of C of x and C of x is basically summation of what, C of x y where y is not y is t minus x . So, this is we are looking for and this we want to show is less than 1 m . So, this is basically we can take the expectation inside. So, this is basically expectation of C of x y and y belongs to; now this is basically 1 by m , this basically 1 by m . So, this is n minus 1 by m this is less than n by m . So, this is the probe. So, expected number of collision is 1 by m . So, that is good. So, expected number

of collision is basically n by m . So, this is the load factor. So, the universal choice is good. So, what is the universal hash function? We have a collection of the hash function along this collection if the bad portion is just 1 by m fraction of the total and then if from this collection total collection if we choose a hash function randomly, uniformly equally likely and if we use the hash function for our hashing and then this hash function is called this choice of a hash function is called universal hash function.

So, in the next class we will construct some universal hash function.

Thank you.