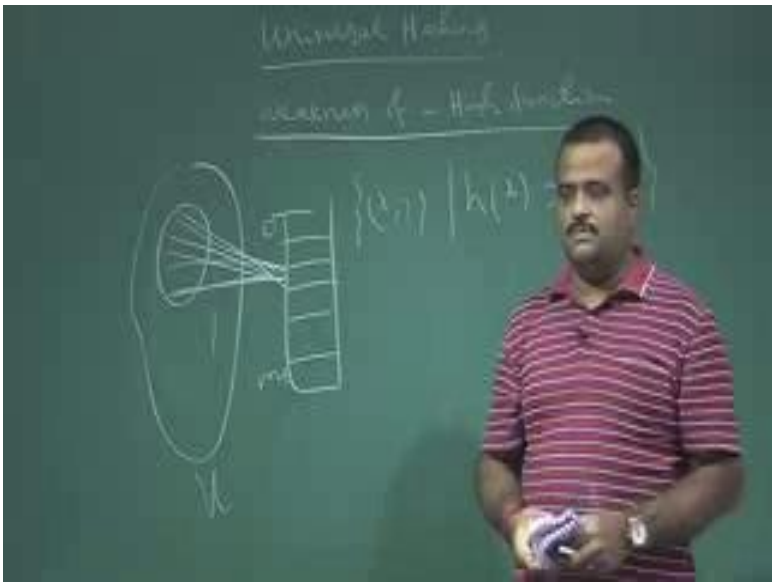
Internetwork Security
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture - 41
Universal Hashing

We will talk about Universal Hashing. So, this is coming from what is the weakness of a hash function.

(Refer Slide Time: 00:25)



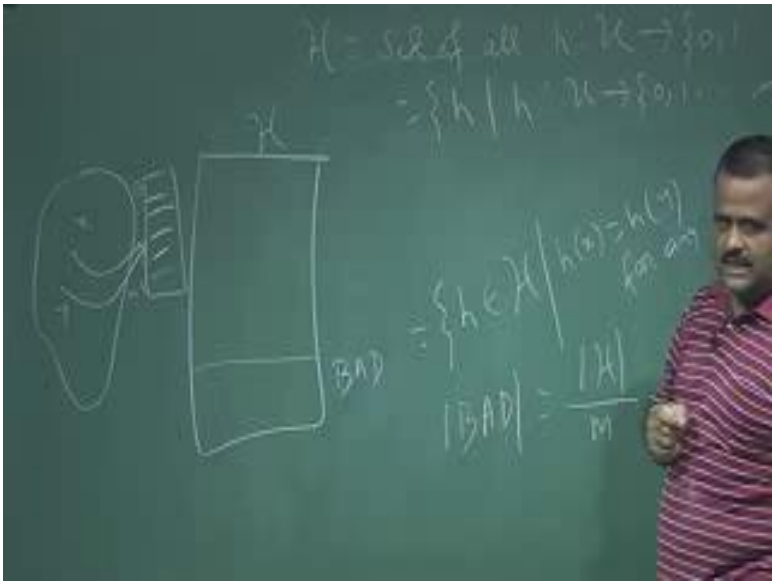
So, this is a fundamental weakness of a hash function. So, hash function is a function from any bit input, we have fixed bit output. So, domain size is very much bigger than the co domain size. So, there has to be collision. So, suppose the some company like Microsoft are they made a competition to get a hash function. So, you develop hash function, your friend develop another hash function and both wants to win the so. So you both submitted your hash function to the Microsoft, then what Microsoft will do Microsoft will give your hash function to your friend and your friend hash function to you and asked to perform, asked to find out for the input where it will perform badly.

So, there has to be certain input where it will perform badly. So, your friend can get such input k or x, y pair such that so this is the; so always it is possible because it is a fixed that co domain size is very less and domain size is more. So, this is your friend can always get a key, a bad portion of the key where everything is mapping to a particular

slot. So, more collisions are there. And can your friend will report to Microsoft, hey this is the set of input where my friends code is gone. I mean it is colliding to the particular slot, so that is always possible if you have time you can do that. So, that is the fundamental weakness of hash functions.

So, to avoid that what we do, we can choose the hash function randomly at the runtime, so that is the idea of choosing the universal hashing. So, in universal hashing, we choose the hash function randomly, randomly at the runtime, so that now what will come with some input where your code is performing badly like it is (Refer Time: 02:53) because you do not know which random variable will be chosen. So, depending on that we will come to the construction of such hash function.

(Refer Slide Time: 03:10)

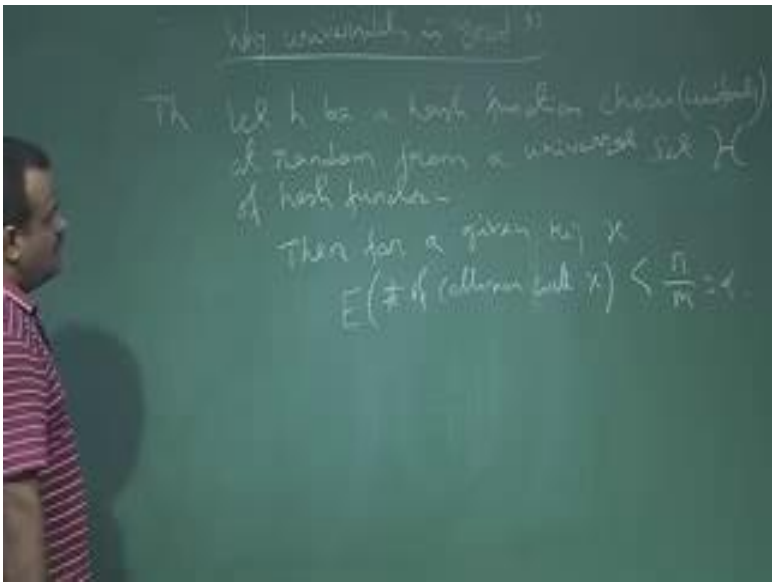


So, let us formally define what is universal hashing. So, suppose this h is a set of all hash functions or collection of hash function, set of all hash function h form u to this. So, the h is this set. So, it is basically set of all hash function which is call u to this. So, this is our collection. Now, among this collection and suppose we have a bad portion. So, bad portion, how we define the bad portion, bad portion means if you take any two key x, y and if we choose the hash function from this bad portion then there has to be then these two will collide. So that means bad portion is set of hash function from this collection of hash function such that h of x will be h of y for any pair of keys, for any x, y belongs to a cross u. So, this is the bad portion, so that means the collision is guaranteed if you

choose our hash function from this then if you take any two keys then they will be colliding.

Now, if the size of this bad portion, if the cardinality or the size of this bad portion is basically size of the total by M that means, M is the slot number of slots, so 0 to m minus 1 is the number of slots. So, if there are 1 by m fraction of the total set is basically bad then we called this collection is universal collection of hash function. And if we choose a hash function randomly from this collection that is universal hashing, and it is a good choice, we will discuss that why it is good. So, this is the universal collection of hash function and why this universality is good that we will talk about.

(Refer Slide Time: 05:46)



So, now the question is why universality is good I mean why this collection is good that means if you take a hash function randomly from this collection why this will perform good. So, this is because if you just write a theorem like this, so let h be a hash function chosen randomly h be a hash function chosen uniformly at random from this form a universal hash universal collection of hash function universal set h of hash function. And then the expected number of collision with x if for a given x then for a given key x which you are searching say we are searching a key.

So then the expected number of collision with this x is less than n by m which is the load factor and that is good. So, if we are searching for key x, now if it is expected number of collision is alpha that means, it is basically the load factor of the table then it is good

news. So, we have to prove this. So, if we choose a hash function uniformly at random from a set of hash function then this is the expected number of collision with x. So, this we need to prove it.

(Refer Slide Time: 08:25)

So, to prove this, we will take help of what is called indicator random variable. Let us denote c x y is 1; if c x y is 1, if h of x equal to h of y, you take two key x, y and we denote this random variable value is 1 if they are colliding 0 – otherwise, this is how we different by indicator random variable. Then what is the probability that c x y is 1, this is basically 1 by m because we have seen if you take any two keys then you are choosing the hash function uniformly at random from this collection. So, it is basically there are total this way we can choose and if we choose that hash function from that bad portion then it will collide and that favourable case is this. So, this is basically 1 by m simple classical definition of probability.

So, this is 1 by m and so what is the expected expectation of this. So, the expectation of this random variable is basically 1 into it is basically by the binomial random variable plus 0 into probability of c x y is equal to 0. So, this is basically 1 by m. Now we are looking for a given x, we are looking for the expected number of key is colliding with x, so that we denote by c of x the number of key is colliding with x. So that is nothing but a c of x y where this y is belongs to key space, I mean the key space which size is n. So, this is basically n minus 1 other than the x.

So, if you take the expectation both sides. So, this is basically expectation of c of x y. So, y is belongs to minus x. So, this is basically n minus 1 by m which is less than m by n alpha. So, this is the proof, so which is basically 1 by m. So, this is good news. So, universally it is useable. So, if we have such collection, where the bad portion is just the 1 by m fraction of the total collection then if we choose a hash function at uniformly at random from this collection then that hash function performance is good; that means, it will distribute the key uniformly; that means, load factor will be preserved. So, next we will talk about how we can construct such a hash function - the universal hash function. So, this is the construction one example of universal hash collection.

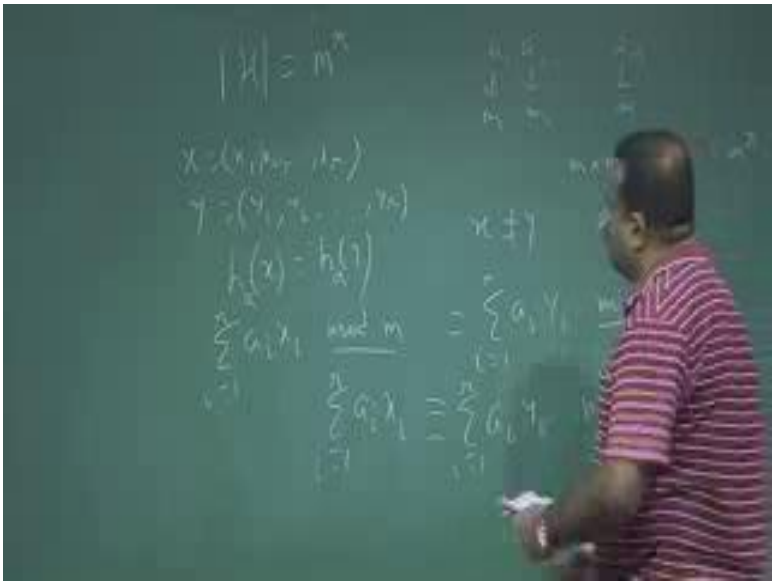
(Refer Slide Time: 12:03)

This is one example of or how to construct the universal example of universal hash function. So, here we divide the keys. So, we have a long keys; we divide the key into r blocks k 1, k 2, k 3, k r such that each of k i's are between 0 and m. So, each of these block size is maximum m, m is the number of slot. So, this is how we divide. So, maybe last block we have paired something to make it m size. So, if m is 2 to the power r or m is 2 to the power say l then each of this block is l bit. So, now, what we do we take a vector - a randomly a 1, a 2, a 3, a r, and this is a i is are also lies between this. And each this a i is chosen randomly uniformly at random. So, this is a constant, we randomly choose this constant, and we fix this constant. We choose this a i is then we have a this vector a and we fix it that is a constant now. Now, we will use this for this hash function

to construct a hash function, and that hash function is denoted by h of a because this will depend on this constant a.

So, this h of a k is basically summation of a i k i i is equal to 1 to r mod m, this is basically the inner product between a vector and this is the key, and this k vector. So, key is basically k 1, k 2, k r and a vector is basically a 1, a 2, a r. So, you are just doing that inner product between a vector and k vector and we obviously have to take the mod m because our slot size is 0 to m minus 1. So, this is one construction of hash function. Now, we need to prove that this collection I mean collection means if you vary a, so a is chosen from this way. So, if you vary a, we are getting different, different hash function. So, varying a means we are varying this a 1, a 2, a r these are chosen randomly. So, this will give us a collection. Basically our collection is h is basically h of a collection, so a is varying, so this collection, so this is chosen randomly. So, this collection we have to prove that this is a universal collection.

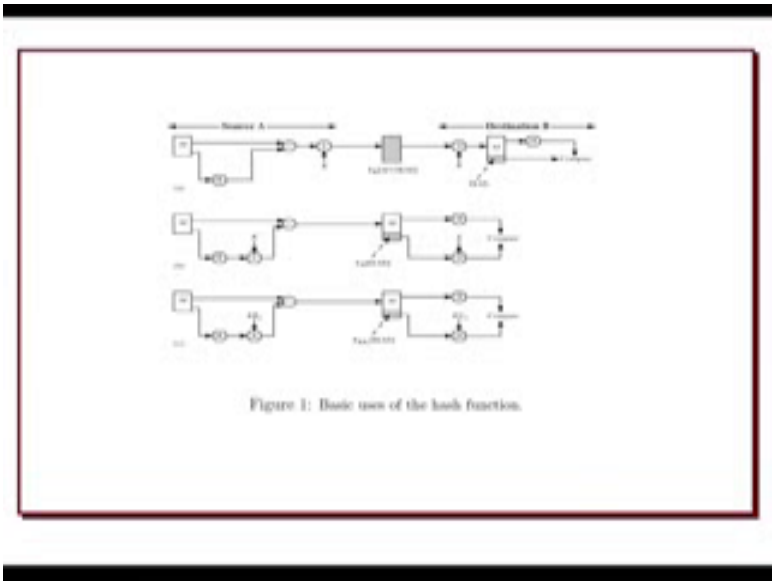
(Refer Slide Time: 15:51)



So, how to prove this? So, what is the cardinality of this h this collection this h is basically. So, we have the choice for a i each a i is are taking value from zero to m minus one. So, this cardinality of this is basically m to the power r, because it is depend on, so this is a 1, a 2, a r. So, each this can be m ways, this can be a m ways like this, this can be m ways. So, m into m into r times, this is basically m to the power r this collection.

So, now, we need to consider the bad portion. So, bad portion means the where coli collision is there so; that means, we take two keys x and y. So, x is x 1, x 2, x r, x 1, x 2, x r dot dot dot; and y is say y 1, y 2, y r. So, now, we want to an x and y is not equal. So, if x, y is not equal you can assume one of this it is not equal. So, without loss of generality, we can say x 1 is not equal to y 1. Now, we take actually we are calculating the cardinality of the bad portion. So, we take h of x equal to h of y. So, h of x is nothing but summation of a i x i mod m is equal to summation of a i, so this is a, this is a, a i y i mod m. So, if these two is under mod m so that means, they are congruent 1 to r i is equal to 1 to r so that means, they are congruent. So that means, a i x i i is equal to one to r is congruent to summation of a i y i, i is equal to 1 to r mod m.

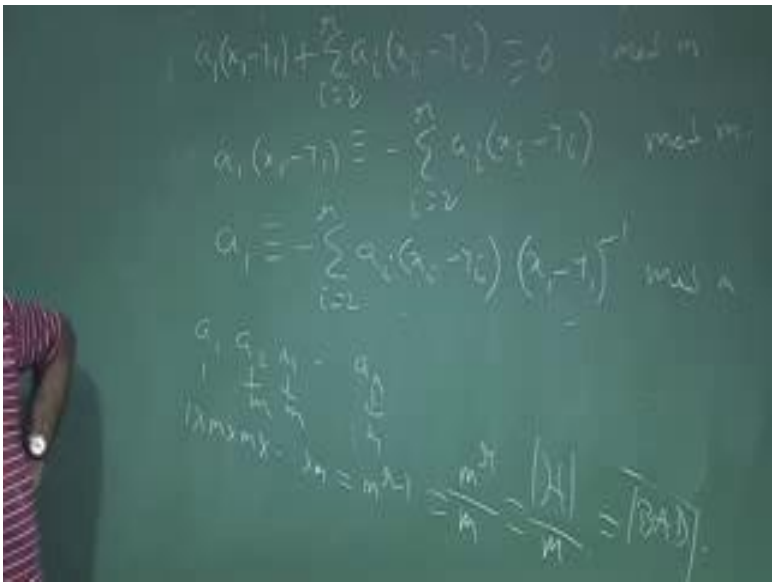
(Refer Slide Time: 18:03)



.

So, now we want to simplify this, we take this here. So, we take this side. So, summation of a 1 we take common x 1 y 1 plus summation of a i x i minus y i, this i is from 2 to r, this is congruent to 0 mod m.

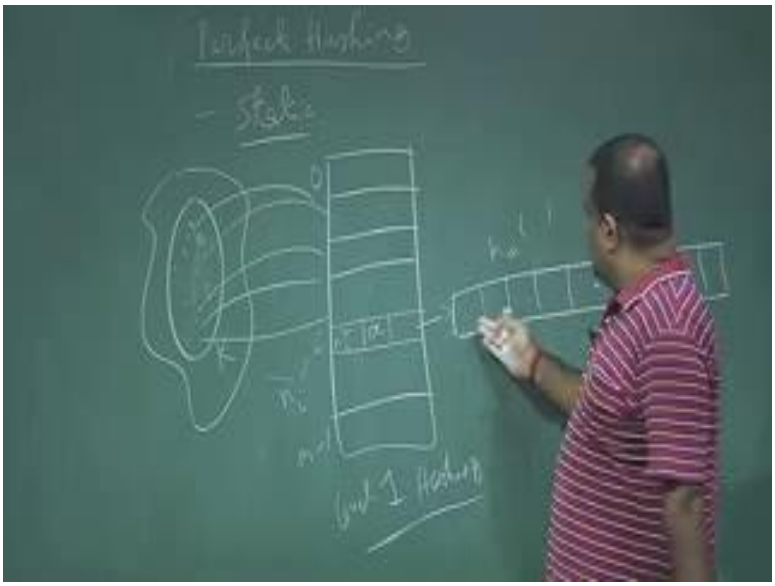
(Refer Slide Time: 18:10).



So, now we take this, this side. So, a 1 x 1 minus y 1 is equal to minus summation of a i x i minus y i i is equal 2 to r mod m. So, now we know this x 1 is not equal to y 1. So, this means x 1 minus 1 y 1 is not equal to 0. So, we can take the inverse of it. So, a 1 is basically is congruent to minus summation of a i x i minus y i i is equal 2 to r into x 1 minus y 1 inverse mod m. So, this will give us no choice for a 1 so that means, so if we have a 1, a 2, a r, so once we choose this then a one is fixed by this relation. So that means, this can be choose a m ways all this can be choose m way, but only thing this can be choose one ways. So, this is total number of such collection is this is the bad collection is 1 into m into m into like this.

So, this is basically m to the power r minus 1, which is basically m to the power r by m. So, this is basically cardinality of h by m. So, this is the size of the bad portion. So, it is the bad portion is 1 by m fraction of the total portion. So, this collection is a universal collection. So, this is an example of universal hashing. Now, if we choose a hash function from this collection that is called universal hashing. Now, let us analyze about the; so this is one construction. Now, let us talk about what is called perfect hashing, there we will use this universal hashing, universal hash function.

(Refer Slide Time: 21:04)



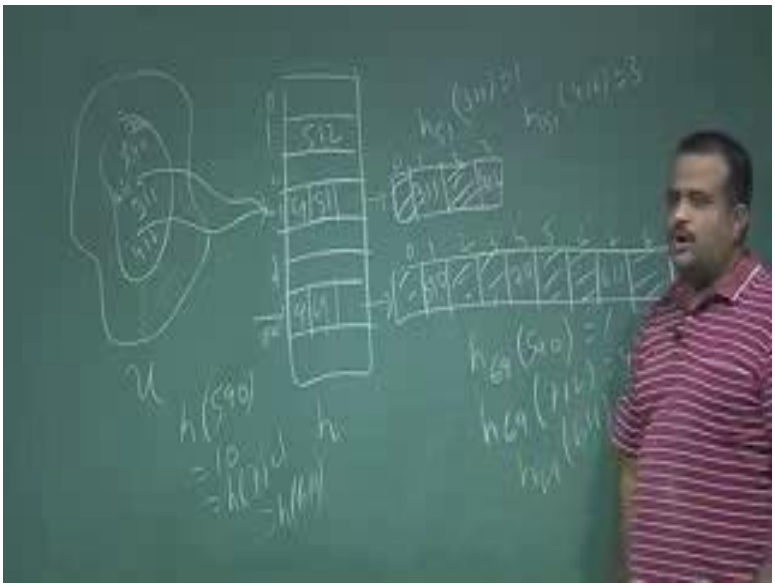
So, in perfect hashing what we have the problem is we have given a set of keys - n keys this is static arrangement. We have given a set of n keys and we need to create the table, so that there will be constant number of I mean the search will be in constant time for searching we should not take more than constant time, so that is the idea. So, this is a static arrangement so that means, we are not allowing any new member to join any new record to be join or any record to be deleted. So, basically we have n keys. So, this is the key pool you have n keys and we have to arrange these in a table in such a way that the searching will be in constant time. So, this is basically a two level hashing.

So, what we are doing. So, in the first level, so there are n keys. So, in the first level, we create a hash function, we choose a hash function this could be a universal hash function. So, we have a original hash function. So, we apply these are the keys. So, we apply this hash function on this keys. This is the first level hash function. This is 0 to m minus 1. This is the level one hashing. So, if we do that there will be collisions. So, suppose at the nth level, so suppose at the ith slot there are n i number of keys are colliding. So, at the ith slot n i number of keys are colliding.

So, what we will do, we will then we cannot store n i keys over here. So, what we do, we will go for the second level hash table. So, here we will have a table which size is n i square, if n i keys are colliding here. And this is seven left the second level hash table. So, this is n i square. And for second level hash function we need to store the hash

function over here, so that we can store by just storing the a vector. And then this is basically h of a we apply on this and which will put it here, which will map this keys into this table.

(Refer Slide Time: 24:02)



So, let us take some example. So, this is the idea. So, this is say we have some setup keys and we have the first level hash function dot dot dot. Now, this is our n keys. Now, suppose here we have some hash file this is only one set, but suppose here in this table, we have two keys which are colliding say 3 1 1, 4 1 2 keys are colliding here say in the first level. So, what we do, we need to create a second level hash function, second level hash table of size, since two keys are colliding of size 4. So, this is 4, this is the size of the next level hash table, and then we need to apply we need to have a value. So, say it could be say 51. So, this is the a value, a will give us the hash function because we need to have the hash function. So, then we apply this h of a. So, this is maybe 3 1 1, this is 4 1 2 and these are empty. So, this is the first level hash function and this is the second level hash function. So, this is 0, 1, 2, 3. So, h of 51 of 3 1 1 is equal to 1, h of 51 of 4 1 2 is equal to 3. But h of 3 1 1 is equal to h of 4 1 2 say this is a 0, 1, 2, 3 slot this is basically 3. So, they are mapping.

Now, suppose we have another slot which is having three keys are colliding then suppose these three keys are say 590, 611, 712, these three keys are say colliding in here. So, in that case, we will put a table of nine slot 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 we put nine slot for the

second level hash function. And you have to have a say this is the 69. So, this is a 590, this is a 712, and this is a 611. So, these are all empty so that means, at the first level all these points are colliding at here. So, say this is set ten slot so that means, h of 590 is equal to 10 is equal to h of 712 is equal to h of 611 these are colliding. So, three-three keys are colliding here. So, this is the now in the first level.

So, for second level, we need to have three square nine table size and we need to choose a hash function that is also uniformly we are choosing from the universal hash function and that a value, a vector is basically we choose some a vector that will give us the hash function. So that means, h of 69 this is the a value of 590 is 1, h of 69 of 712 is equal to 4, h of 69 of 611 is equal to 7, sorry 9 slot. So, it should be 0 to 8. So, this is the way we just have the second level hashing.

(Refer Slide Time: 28:31)

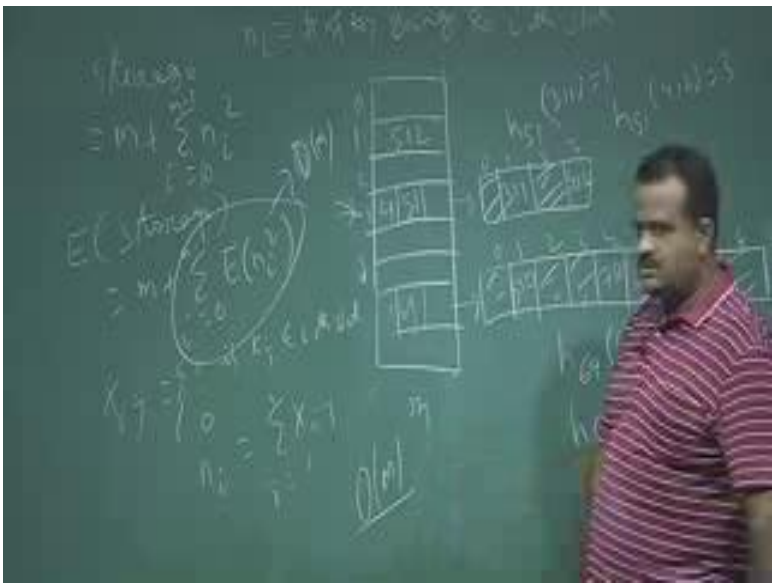
So, in the second level the collision we can guarantee that is nil because suppose we have three keys and we are having nine slot. So, three keys are distributing in the nine slot. So, if we have n keys we are distributing in the n square slot then the number of collision is even less than one that we can prove. So, we have huge space to distribute the keys. So, the collision will be less in the second level. We can guarantee that on an expected second level there will be no collision even less than half collision will be there.

So, how to prove that. So, suppose this theorem is telling, suppose there are n keys and there are n slots at distributing, this is in the second level analysis then the collision is

number of collisions should be less than half we want to prove. So, then if we take two pair of keys, what is the chances of they will collide, the probability of collision is basically the probability of collision is 1 by m this is because we are choosing uniformly. Now, there are n c 2 number of pairs. So, this is the expected number of collisions. So, this is basically n into n minus 1 by 2 in n square. So, this is less than half.

So, even we cannot expect one collision. So, expected number of collision even is less than half, because here we have nine slot and we are storing only three keys. So, collision is nil actually. If our hash function is uniform hash function, it will distribute the key uniform among the slot. Now, another thing we should analyze the storage; intuitively it is taking the we are taking huge number of storage in the second level. In the first level, storage is m, size is m. And in the second level, intuitively we are having the n square i mean if the n i is the number of keys colliding in the ith slot, so we will do the storage analysis.

(Refer Slide Time: 30:06)



Storage is basically m for the first level, and the second level is n i square; i is equal to we have 0 to m minus 1. So, this is basically n i is basically number of keys going to colliding to ith slot. Then if you take the expectation of this storage then it is basically m plus summation of expectation of n i square. Now, these analysis we can this is the similar analysis if you are familiar with some sorting algorithm, which is called bucket sort, if we have n keys we are distributing into the buckets. And if if we denote this x i j

is basically 1, if the k j is going to this slot - ith slot or ith bucket, 0 otherwise then the basically n i is basically summation of x i j, j is coming from how many keys are there want to n something like that. So, squaring means squaring of this if you take the expectation this can be shown that, this is also order of m. This analysis is there in the any algorithm book, this is the bucket slot analysis. So, this storage is basically again order of m. So, we are not really using the extra memory much here this is reasonable order of m.

Thank you.