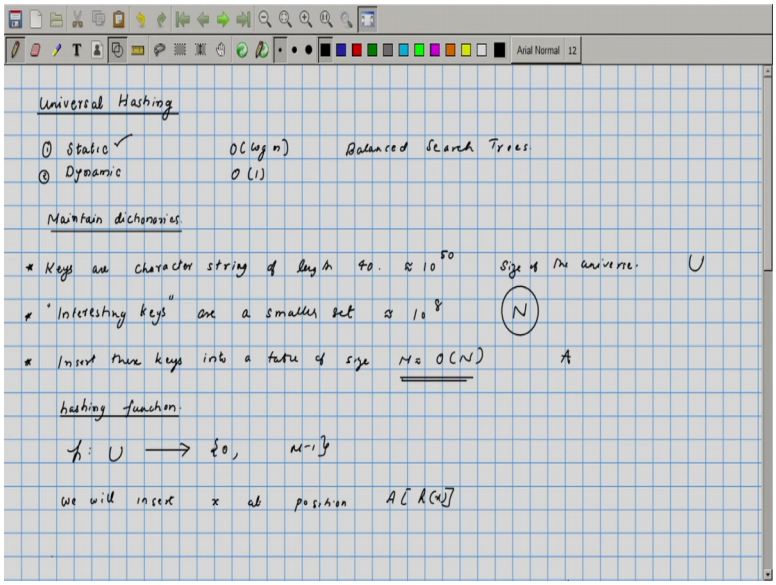
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 35
Hashing

(Refer Slide Time: 00:29)



In today's lecture, we will learn about universal hashing. Universal hashing was the technique by which we could do hashing and the hashing when done in this manner gives provable guarantees about the expected running time of programs. So, let us understand what the basic question is. So, we had looked at all the basic data structure in problem.

So, typically there are two kinds of data structure in problem. One is the static data structure in problem whereas, the other is the dynamic. In these data structure in problems, we have to essentially maintain dictionaries ok, which can answer queries about whether an element is present or not. So, that would have been the static one, but the dictionaries keep changing with time learns the dynamic data structure in problem. There would be insertions, deletions and other kind of updation operations in dictionaries.

The basic operations are insert, delete and find. So, these operations we want to do efficiently and we know that there is a O log n O log n would be in the worst case, deterministic algorithms performance guarantees.

So, we can design by means of let us say balanced trees. We can do all these operations in worst case O log n; where n has the size of the universe or you can think of n is the total number of operations that we are doing. So, per operation we are taking log n. We are interested in knowing whether we can do these in O 1 ok. So, we will see that we can do this in an expected sense if we use hash tables. So, we will assume that say the keys that we are inserting into these data structures. Let us as a concrete example say that the keys are let us say character strings of length 40.

So, suppose we knew that our universe basically contains of names of people or customers we may assume that each that is a 40 length string. So, approximately you can say that this is about 10 to the 50 little more than 10 to the power 50. So, these many different elements make our universe. So, this is the size of the universe ok. But typically our set of interest might be much smaller than that ok. So, let say that the interesting keys of far queue are so, let us say their size something like say 1 millionaire say 10 to the may be little more the 10 to the power 8 and if you compare the interesting keys with the total possibilities size of the universe; size of the universe is enormous ok.

And let us call this by so, universe will denote by U and its sizes also let say we call it as U and the interesting keys let us say its size is N ok. We will want to store this in a table and the table should essentially provide us with constant time operations ok. So, the basic idea is insert these keys into a table of size let us say M ok. So, M should be approximately O of n may be some 10 times and something like that.

The question is where exactly in the table do we insert. We could assume that the keys are all coming from a sorted universe ok. So, these keys where exactly do we insert in the table is a question. We want to insert in such a way that we can do all our operations in constant time. So, the basic idea is to hash.

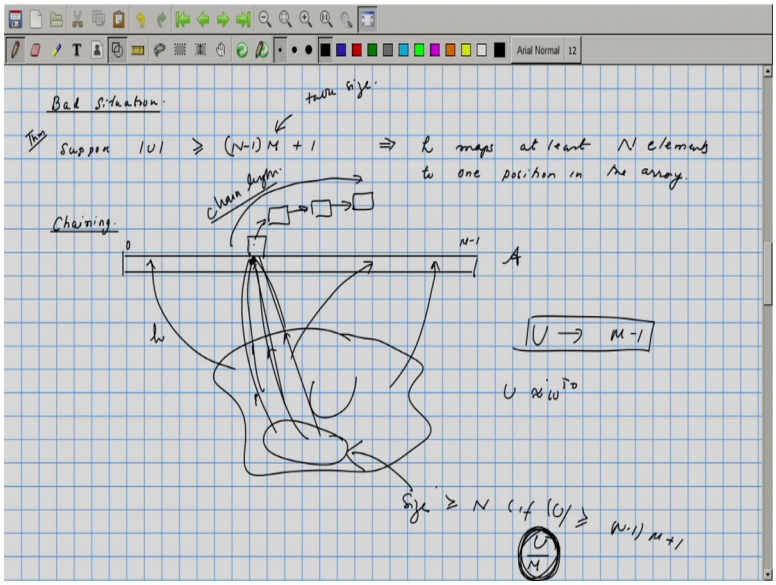
So, let us see what is the hashing function. So, you can think of it as a function h from U to let us say 0 to M minus 1. In other words given any element of the universe we are told at which position in the table which we are going to index by 0 to M minus 1 which is a position where the element x would be inserted. So, if we call the table as A we will insert x at position A of h of x.

Now, obviously, there lot of pressing questions; what if multiple elements not to the same position. We had assumed that N is much smaller than let us say the total size of the

universe and M is let us say something like 10 times n ok. So, even if we assume that the table is the size 10 to the power 10, we know that there will be some particular positions which could contain 10 to the 40 different elements. So, that is not going to be a good case ok.

So, let us first understand what exactly is the issue.

(Refer Slide Time: 06:49)



So, bad situation let us. So, suppose the size of the universe is greater than N minus 1 times M plus 1 ok; M is the table size. This would imply that h maps at least this is for any h, N elements to one position in the array ok. So, this is going to be called as collision. So, there is going to be one particular position where there is large number of collisions. How do we address these collisions?

So, a simple technique is called as chaining ok. So, we could think of this as the array a containing positions 0 to M minus 1 and this is our universe ok. So, each element will via the function h mapped some particular position when the above theorem or lemma would essentially mean that there is some position where lot of elements would mapped ok. And these could be of size greater than or equal to N ok, if the universe is greater than size N minus 1 times M plus 1 ok. So, at least mean. So, you can think of it as a U by M I mean elements would essentially be mapping to there will be one position which contains U by M elements more than U by M elements ok.

So, in this case what we will do is the first element that maps, we will write it there and then we will think of this as a link list starting from there. So, all the elements that mapped to the same position we will just link it and keep it there. So, this is called as chaining. So, now, if any insertions is to be done we can just go to the start or end of the list and inserted there. If there is a fined, then probably it is going to take more time. So, the time taken for the operations is going to be proportional to the length of the chain ok. So, if the chain length is large. When is the chain length large? When there lots of collisions ok.

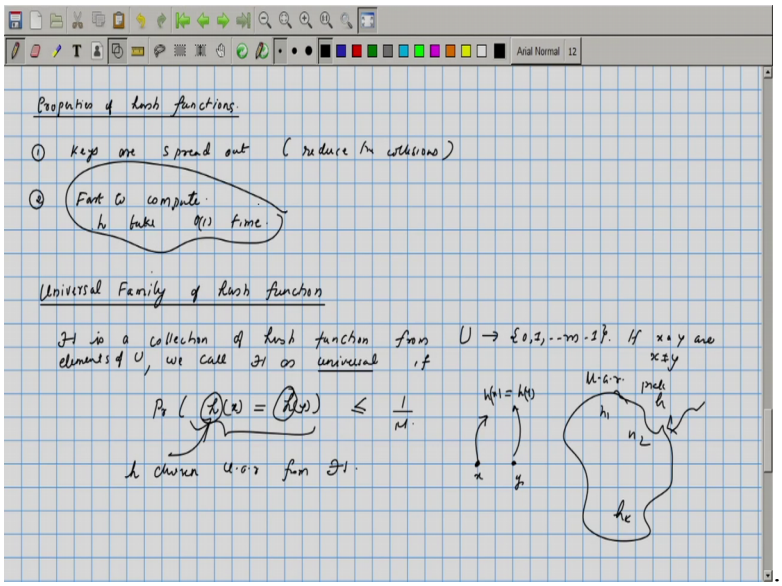
So, in those cases the running time becomes large. It could be as large as U by M and U being a very large collection U by M and M being small U by M is not going to be; I mean there are so, many elements mapping to the same position. There is it looks like there is no escape. But here what we will see is that randomisation helps quite a bit ok.

How does randomisation help? As an overview randomisation helps in the same way. Randomisation is of help in the case of quick sort algorithm. So, if you recall in quick sort algorithm we were choosing pivots randomly. We could think of let us say the pivots being chosen deterministically and once the code is available the adversary can look at the code and based on the choices of pivots made which is deterministic and hence known ahead of time even before the code is run, the choices are clear. So, because of that the adversary can devise a certain input sequence which will make the algorithm perform particularly full.

So, the way out was if the choice of pivot was random the adversary could not design an input which is bound to take large amount of time while quick sort is being run. So, random pivots in that sense spoils the adversaries strategies of choosing the input level. We will get the same advantage by randomness ok.

What we will do is instead of choosing deterministic hash functions which is bound to have let us say pathological configurations ok. If take a deterministic hash functions its guaranteed that there are going to be large number of elements which map to the same position ok. Whereas, if we randomly choose the hash function, the adversary cannot choose a particular dictionary or set of elements which could make the algorithm perform poorly. So, we will see this I mean how this works. We will introduce the notion of universal hash functions.

(Refer Slide Time: 12:37)



So, first of all look at the properties that we will require a has functions and later on we will define what are universal functions. The first requirement would be that the keys are spread out ok. So, hash function basically takes the element of the universe and maps it to some position in the table ok. So, we do not want to many collisions. So, that is one of the requirements; reduce the collisions ok.

The second thing this should be fast compute. If we had randomly chosen our hash function that is each element will map to a random position in 0 to M minus 1, then we can say that chances of collision is going to be very small. But the problem is if we were to choose one such random function, we need to store that entire function which is a map from the universe to from U to M minus 1 that is going to be a very large object in itself. If U was say something like 10 to the power 50 we need to store where exactly each of these 10 to the power 50 elements mapped. So, storing that table is going to be more expensive than actually maintaining the dictionary ok. So, this should be fast compute. So, we will require and we will just say that the hash functions takes O of one time the take constant time to compute. So, these are the basic requirements of hash functions ok.

And now let us define what are called as universal family of hash functions. Some textbook would refer to what we define as universal family has 2 uniform family. There is more general notion of k uniform family as well ok. So, let say h is a collection of hash functions from say U to m minus 1 say if x and y are elements of U, we call h as

universal or universal family of hash function, if the probability of h x being equal to h y is less than 1 by M.

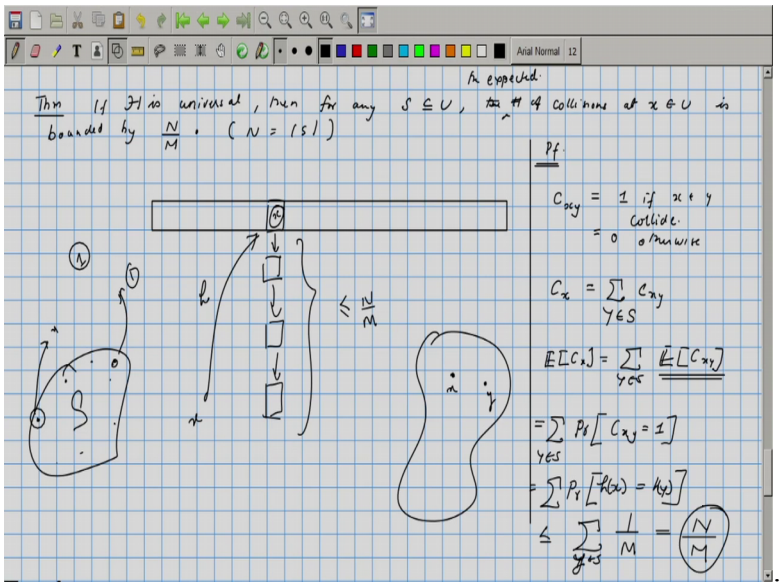
So, what is this probability; I mean what is this event? We are fixing some particular x and y and this should be true for all particular all choices of x and y where x not equal to y ok. So, we will x is not equal to y because if x was equal to y then hx is of course, equal to hy ok. So, probability that hx is equal to hy when x is not equal to y should be less than 1 by M. Now the probability O is over choices of h ok; so, h chosen uniformly at random from script h. So, we have a bang of hash functions let us say h 1 h 2 h k and we have some 2 fixed elements x and y or somebody gives you some arbitrary elements x and y ok.

Now, for every such choice of x and y such that x and y is not equal if we randomly choose. So, uniformly at random pick h and then apply h to x you will get a checks and apply h to y you will get hy. Now this could be equal probability of that should be less than 1 by M. If that condition is met then this particular collection of hash functions is called as a universal family of hash functions. So, this just provides a definition of what is universal family. We need to worry about whether we can actually construct such families.

We will see that there we can construct such families where takes O of 1 time to compute h x for any particular x ok. So, we will also see that once we can define such universal families and once we show the existence of such families we will see that we can do dictionary operations in expected constant type ok.

So, first let us see I mean why universal families are helpful I mean universal family of hash functions are helpful ok.

(Refer Slide Time: 18:15)



So, first theorem: so, this theorem states that if you have universal families and there are no bad inputs. So, what is an input here ok? When we are thinking of dictionary operations, the set of interesting case is what we will be whether input ok. So, look at any sequence of inserts and deletes they are over some particular subset of the universe whose size is subset is that is bounded by M.

So, you take any subset of this size we will show that we have a universal family then the insertions and deletions basically happen in constant time. But before we show that we will show that there are no bad subsets in the sense the number of collisions is going to be small. So, if h is universal then for any s subset of U the number of collisions at x; that means, if you pick any element x belong in to universe the number of elements that could collide with x is bounded by N over M. So, N being the size of s simple theorem, but it is a very useful one this states that for any element of the universe. We were our hashing function basically was randomly chosen hash function let us call it as h.

Now, an element x would map to the position hx and there could be a chain starting at that position. What we want to look at is the length of the chain this is less then N by M that is what we are saying the expected value is less N by M ok. So, let us see y. So, we should say the number of the expected number of collisions ok. So, proof is straightforward.

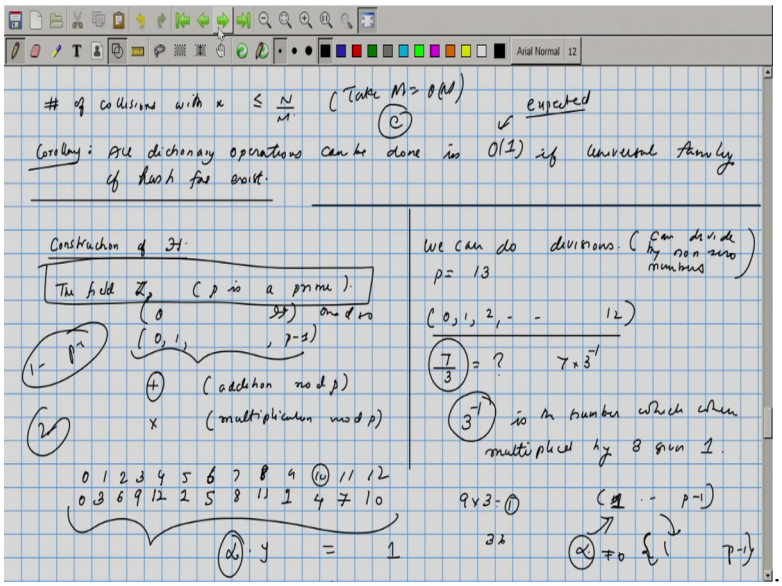
So, let us say that C xy is the random variable which takes the value 1 if x and y collide ok. So, x and y are to run I mean 2 arbitrary values inside the universe. So, this universe had some let us say x and y are 2 arbitrary elements. For each of those we will have a random variables C xy it takes the value 1 if the random function which we took or the random hash function we took mapped x and y to the same location then we will say that C xy equals 1 and it is equal to 0 otherwise ok. So, C x defined as some over all y C xy this is going to be the number of. So, here y should belong to yes. So, look at yes. Each element of s can collide with any with an element x

So, our set x was there we want to know how many of these would collide with x that would be nothing but summation over y belonging to yes. So, we could look at all elements inside this set. Does this particular element collide with x? If it does we will get a one if this element collides we will get another one and so on. If you add up all these what you get is C x. So, C x is the number of collisions involved at position x when we use a random hash function from h ok.

So, the expectation of C x is nothing but summation over y, expectation of C xy and expectation of C xy; C xy being an indicator random variable its expectation is nothing, but the probability that C x y equals 1. So, this is equal to sum over y belonging to yes probability that C xy is equal to 1 that is same as the probability that x and y collide. So, x and y collide means hx is equal to hy and we know that for any 2 element x and y probability that they collide is less than 1 by M. So, this summation y belonging to yes 1 by M which is going to be equal to N, the number of y I mean elements inside yes that is N. So, this is N by M.

So, if you have a universal family the number of collisions at any point x is at most N by M.

(Refer Slide Time: 23:45)



So, that is our take home message. Number of collisions with x; this x could be any element inside the universe is going to be less than N by M and as a corollary we can claim that all dictionary operations can be done in O 1 if universal. So, this is in an expected sense if universal family of hash functions exist that is true just because the time taken for any operation is going to be proportional to the number of collisions and if we take let us say M to be take M to be O of N and this entire expression becomes let us say some constant N divided by O of; so, some constant. So, we know that every dictionary operation in insert delete find etcetera can be done in linear O of 1 time.

So, now the next topic would be this is possible if we have a universal family, but how do we construct a universal family of hash functions ok? So, we will just need small recap about the field of integers mod p ok. So, what is this? If you look at the collection of numbers from 1 to p minus 1 and we define the addition as addition mod p and multiplication as multiplication mod p then this algebraic structure has some interesting properties ok.

So, one is we can do divisions ok. So, when we say we can do divisions means we can divide by nonzero elements ok. So, I mean let us take an example. So, if p is equal to let us say 13 ok, the collection of numbers that we are interested in 0 1 2 up to 12 ok. Somebody asked is what is let us say, 7 by 3 ok. If you had carried out this in let us say the field of real numbers then you will get some answer, but here we want the answer to

this question what is 7 divided by 3 to belong to this collection, so that it satisfies all the familiar rules of arithmetic ok.

So, we can write 7 as 7 into 3 inverse and 3 inverse is going to be that particular element which when multiplied by 3 gives us 1 ok. So, 7 divided by 3 we will simply view it as 7 into 3 inverse and 3 inverse is the number which when multiplied by 3 gives 1. Now is there any element in this collection when multiplied by 3 gives 1? Well, we can just multiply and see. 0 into 3 is going to be 0 1 into 3 is going to be 3 2 into 3 is 6 3 into 3 is 9 4 into 3 is 12 5 into 3 15, 15 mod 13 is 2 18 mod 13 as 5 21 mod 13 as 8 24 mod 13 as 11 20 7 mod 13 as 1 ok.

So, we know that 9 into 3 is equal to 1 ok, maybe there other elements. 30 in to 3 is 30 mod 13 as 4 7 36 as 10 ok. What you can observe is that all these elements are distinct ok. When you had multiplied the collection 0 to 12 with the element 3 we got all elements in this collection from 0 to 12 and therefore, one should of course, be present if you say that all elements from 0 to 12 are present 1 is also present and that is what is true when p is a prime number ok.

If you take this collection 0 to p minus 1 and multiplied it with any alpha in 0 to I mean let say alpha should not be equal to 0 ok, if you multiply these collections or maybe I will just start with 1 to p minus 1 and multiplied it with any alpha which is not equal to 0 the result will be again 1 to p minus 1 in some other order ok. And therefore, you can say that every element alpha multiplied by y if you vary alpha or everything then you are going to get 1 for some particular alpha and that alpha is a unique alpha that is going to be the inverse of y. This is true when you have this collection as 1 to p minus 1 ok.

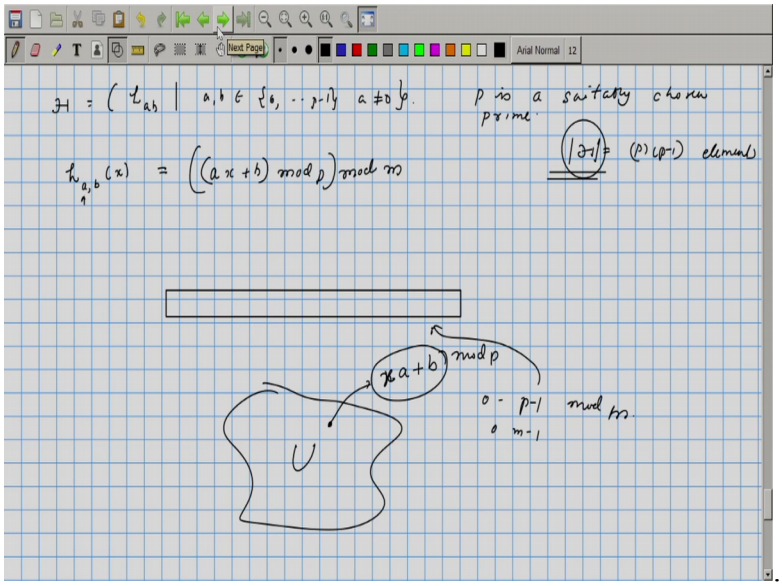
In other words this is the prove that can be made more formal that every element has an inverse ok. So, you can do addition and additive inverse clearly is there with any element you can add another elements so that you will get 0 mod p. And multiplicative inverses are also there means for every element you can multiply with the some other element so that you get 1 ok.

So, since additive inverse is an multiplicative inverse are present we can essentially do arithmetic all kinds of arithmetic ok. Any division would essentially be translated will be thought of as multiplication by the inverse ok. And this is clearly not true when this collection is not a prime mean for example, if you take an 0 to 99 and if you your

operations are mod 100. If you take 2 you multiply 2 with any other element and do this multiplication mode 100 the resulting number will be even in particular we will not get 1 ok.

So, this existence of inverse is a characteristic of the and this happens while you are looking at fields. In every element we will have a multiplicative inverse.

(Refer Slide Time: 31:45)



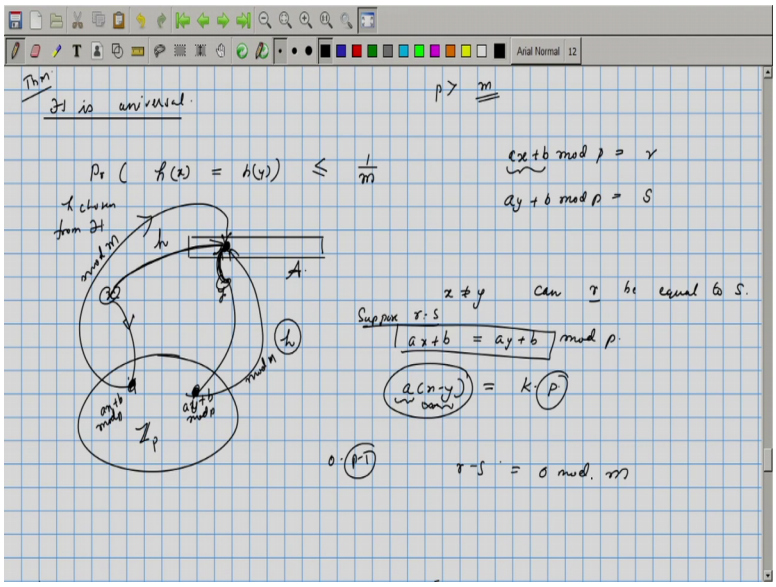
So, what we will do is we will first describe the family h and then we will show that that is universal. So, h will basically consist of functions of the form h a comma b ok. So, its two parameter family of functions is going to be each function in h is going to be characterised by an a and b ok. So, a b belongs to 0 p minus 1 a not equal to 0 and h a b of x is nothing but say a x plus b mod p. So, here p is a prime number which are suitably chosen and h a b comma x is x plus b mod p the whole mod m.

So, what we do is for any element of the universe suppose this is our universe we will first compute x times a plus b mod p ok. So, this will result in a number between 0 to p minus 1 and that number is basically mapped into the array by doing mod Mm ok. So, that maps it to some locations 0 to m minus 1 ok.

So, look at all such functions where a is not equal to 0. So, h basically consist of p into p minus 1 elements ok; size of h is this much and each time we run our algorithm what we are essentially doing is we are choosing one random element of this function ok. So,

once we have fixed our dictionary scheme we will choose a random input I mean we will choose a random function and then computed for that and that random function remains throughout the execution ok.

(Refer Slide Time: 34:19)



So, now we need to look at what is the; I mean why is this universal ok. So, we need to prove. So, h is universal is our theorem which means what we have to prove is probability that h x is equal to hy; h chosen from script h this we have to show this is going to be less than 1 by size of m ok. So, let us call that as small m ok. How do we show this?

So, let us fix some particular x and y ok. So, first these gets mapped to I mean. So, we choose a random h and we want to compute the probability the random h would map these 2 elements to the same position in the array a. There were m positions that x could have been mapped to, but we are not choosing a random position from this I mean from this array, but instead we are first mapping x to x plus b mod p and this we are going to mapped to A y plus b mod p and this mod n wherever I mean mod n it will take into this position and mod N it will the I mean sorry mod m will take you to this position mod m would take you to again this position ok.

So, whatever we are doing in these two steps; first to Z p and then to A, we can think of it as x mapping indirectly to this particular position. So, this is our function h. So, h is

essentially composition of two functions and what does the probability that x maps two things to the same position ok.

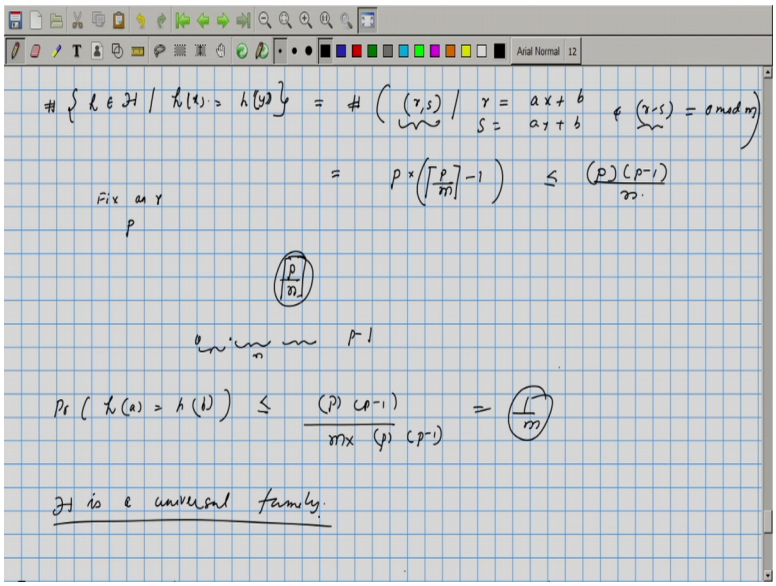
So, for x let us call its image in Z p as r. So, a x plus b mod p let us call it as r and a y plus b mod p is equal to r where p we will assume p is greater than m ok. Now this a x plus b if x is not equal to y can these 2 elements; so, let us assume that x is not equal to y can ok; so, x; so, sorry this type here. So, a y plus b we call it as s is not equal to y. Can r b equal to S? Let us see if it can be. So, then a x plus b is equal to a y plus b mod p.

Suppose r is equal to s this condition is true. That would imply that a times x minus y is equal to k times p ok. Now a is chosen from 0 to p minus 1 and x minus y also we will ensure that it is less than p. Since these are two elements which are less than p cannot mean divide either of them and p being a prime must divide one of them ok.

So, if we chose p to be larger than the let us say the maximum values that x and y could take and such a prime surely exist, if you could take such a prime then this does not happen that is a x plus b will not be equal to a y plus b mod p. And therefore, we can assume that r when you choose x and y as unequal elements ax plus b and ay plus b are also going to be unequal ok.

Now, what is the probability the two such elements will map to the same position that is what we need to compute. So, intuition is simple; distinct elements x and y mapped to distinct positions, but x and y could have been adversarial. But now since we have randomised means since a and I mean a and b are being chosen randomly this basic is a random element of Z ok.

(Refer Slide Time: 39:29)



Lets compute the probabilities. We need to compute the size of the following site h belonging to script h such that hx is equal to hy; for some arbitrary element x and y the number of functions which would map x and y to the same element. So, this we can think of as this is equal to the size of r comma s pairs such that r is equal to let say a x plus b and s is equal to a y plus b and r minus s is equal to 0 mod n.

So, all those r and s so, here when x and y are different r and s are going to be different. So, two r and s will map to the same position only if r mod r minus s is going to be 0 mod n ok. So, we need to compute the total number of such pairs ok. If you fix an r ok; so, r could be chosen in p ways r could be any number between 0 to p minus 1 and for each s for a fixed r how many s's are there which satisfies this equation; r minus s is equal to 0 mod N at most p by n ok; s could have been p of them ok.

So, 0 to p minus 1 all those ones which are separated by distance n could fall in the same equivalence class. So, that can be at most let us say ceiling of p minus 1. So, the size of this set is bounded by p, p is for the choice of r and s there can be at most p divided by n minus 1 because r equal to s is something that we do not want and this is surely less than equal to p into p minus 1 by n.

So, we have basically modified some notation; mean n is m that is size of the table ok. So, we were looking at a x plus b mod p mod m ok. So, if r is not equal to s they mapped to the same position only if r minus s is equal to 0 mod m ok. So, this is going to be 0

mod m and the total number of such pairs is less than p into p divided by m ceiling of that minus 1. So, that is less than p into p minus 1 by n.

Therefore the probability that ha is equal to hb is going to be less than this number that is p into to p minus 1 by N divided by total choices of h; the total number of h where p into p minus 1. So, this is equal to 1 by m ok. So, with this we can conclude that h is going to be a universal family of hash functions ok. So, basically what it means is we can do all the dictionary operations in constant time ok. We will stop here for today.