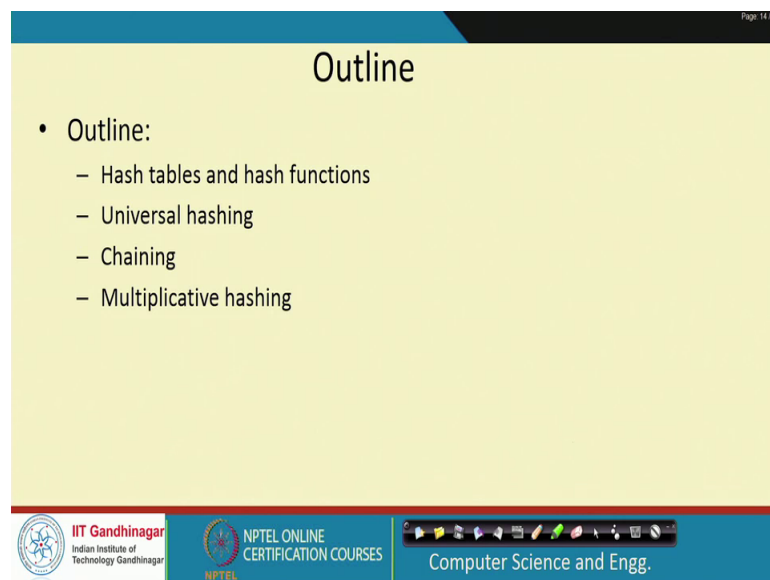


Scalable Data Science
Prof. Anirban Dasgupta
Department of Computer Science and Engineering
Indian Institute of Technology, Gandhinagar

Lecture - 06
Introduction to Hashing

Hello everybody welcome to the course of scaling or Scalable Data Science. Today's lecture is on introduction to Hashing, my name is Anirban Dasgupta, I am a Professor of Computer Science and Engineering at IIT Gandhinagar, so let us begin.

(Refer Slide Time: 00:33)



Page 14/14

Outline

- Outline:
 - Hash tables and hash functions
 - Universal hashing
 - Chaining
 - Multiplicative hashing

IIT Gandhinagar
Indian Institute of Technology Gandhinagar

NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

Computer Science and Engg.

The outline of today's lecture will involve looking at Hash tables and Hash functions. While this might be something that you have touched upon in your under graduate algorithms class will go in to at little in the little more details. Will first see that what makes a good hash table, what makes a good hash function right and in order to define this well will introduce a concept of universal hashing. Then will look at some analysis of hash function or a hash table in terms of the query time right and the concept of chaining will come in here. We will then look at how to construct good hash tables and in this lecture itself will look at one very simple example of simple construction for a good hash function.

(Refer Slide Time: 01:31)

The slide is titled "Querying" and features a large image of a library on the left. On the right, there is a smaller image of a book titled "Game of Thrones" by George R. R. Martin. A blue arrow points from the book towards the library, with the word "Present?" written above it. Below the arrow, the text reads "Naïve algorithm: linear in dataset size". The slide includes logos for IIT Gandhinagar and NPTEL Online Certification Courses at the bottom, along with a small video inset of a speaker.

So, here is a basic problem, suppose you have a library right that has a huge number of books and you have organized it in whatever manner you can. Now somebody comes to you and gives you one particular book and you have to ask the question of whether this book is present in your library or not ok. So, what is a Naïve algorithm for this the most naive algorithm would be a linear search through your entire data sets ok.

We can always do that, but can we do better than this, can we pre process the data that we have to build a data structure. So, that we are answering the query of that given any particular element we are answering the query of whether it belongs to my data set in time that is much faster than going to the entire dataset and for this will construct a hash table.

(Refer Slide Time: 02:28)

The slide is titled "Hash Table" and contains the following content:

- Elements come from universe U , but we need to store only n items, $n < |U|$
- Hash table
 - array of size m
 - Hash function $h: U \rightarrow \{0, 1, \dots, m-1\}$
- We typically use $m \ll |U|$ as well as $m < n$
 - Collisions happen when $x \neq y$, but $h(x) = h(y)$

The slide footer includes the IIT Gandhinagar logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a speaker.

The hash table is at its start a very simple data structure, we start by assuming that elements come from a universe U right think of this as let us say the titles of all the books or the eyes bin numbers of all the books. But while the, but while the universe U is large at any point we need to store only n items small n items and small n is potentially much less than the cardinality of the entire set U . Our hash table data structure will consist of 2 parts, one this is simple array of size m right.

So, this the size that we choose m here the size m that we choose here will be bounded by or accessed to memory, how much memory to be have how bit can be to make the hash table. The next is the most important part of a hash function of a hash table, a hash function and this is a function that maps every element of U into a number from 0 to m minus 1. So, think of a hash function as telling at the following, given the element x in which position of the array should you place the element x right. So, h of x will give you the position of the array in which they place element x ok.

Again we will typically use m that is much smaller than the cardinality of the of the of the universe, m will also be much smaller than the typically be much smaller than the than the size of the elements that you need to get right and because of this because m is less than n collisions about to happen right, by something if you have read about it called the pigeonhole principle.

So, I mean so collisions we call it a collision when for 2 elements x and y of the universe x does not equal y , but h is asking them to map into the same position of the array. That is h is telling you that h of x is equal to h of y and so it should be put in both of this in the same position of the array and we call this a collision.

(Refer Slide Time: 04:42)

The slide is titled "Hash functions" and contains the following content:

- In theory, we design for worst-case behaviour of data
 - Need to choose hash function “randomly”
- Hash family $H = \{h_1, h_2, \dots\}$
 - When creating hash table, a **single** function $h \in H$ is chosen randomly
 - We then analyse the expected query time
- Since the algo has to carry around the “description” of the hash function, it needs $\log(|H|)$ bits of storage
 - $|H|$ cannot too big, in particular, it cannot be the set $[m]^U$ all possible functions

Handwritten annotations on the slide include a blue circle around the H in the second bullet point, a blue circle around $[m]^U$ in the third bullet point, and the handwritten text $1/\log m$ next to the word "expected" in the second bullet point.

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 speaker.

So, why do we look at collisions right, in theory whatever looking to do right, we are looking to do what we are look in to do is keep all the elements x in the in the same position h of x right. So, therefore if there is collision then in some sense the elements are lamping up at the same position in at the same place of the hash table right. So, what we want is to kind of distribute it distribute the elements of the universe as uniformly across the hash, as uniformly across the area is possible right and it is clear that you cannot do this always. Suppose so consider this, suppose you came up with the deterministic hash function and you showed this to your advisory right.

Now, by the pigeonhole principle this deterministic hash function H (Refer Time: 05:45) of greater maps a lot of elements into the same position into one into one place of the hash table right that is a position 0. So, in adversity could now give you a data could now give you a dataset right that consists exactly of those elements of the universe right. That is basically what it does it takes h inverse of 0 and gives you that positive dataset right. So, then anything that you do has to boil down to just naive search over these elements ok.

So in some sense we cannot really guarantee anything if you are choosing your hash function in a deterministic manner and if you are advisory is powerful to sort of look at this hash function and design the dataset and therefore what we deserve to is whereas known as a randomized algorithm that is what we say is that ok. The hash function will not be chosen deterministically, but it will be chosen randomly reasonably from a family of functions ok. So, the most ideal way to choose a hash function seems to be choosing a function at random that is out of all possible functions ok.

So, the so in that case the family H that we are talking about could potentially be the family of all the functions that go from U to m right, that go from U to the set 0 to m minus 1 and at one time what the algorithm is going to do is that it is its going to pick a single hash function out of this set, out of this family H and what we will then do is analyze the expected query time of the algorithm. So, this expectation that we take this expectation that we take here is really over the choice of the hash function it is very important to remember that, that the algorithm's random choice right is the choice of the hash function and the analysis of the algorithm happens with respect to the this randomness. So, it might seem natural to choose the hash function uniformly at random out of all possible functions.

However, there is a catch here right and this catch is as follows that, because the algorithm has to also carry around the description of the hash function right. Because remember when the query comes algorithm again has to calculate the value of the of the h of the query right. So, therefore it has to carry around code that is essentially the description of the hash function with itself. So, therefore in an information theoretic sense if the size of the family is H is cardinality of H right, the algorithm has to carry logarithmic number of bits in the cardinality of H as the description of the hash function right and if the set H is the is the is the is the set of all the hash functions m to that maps U to the set m right.

Then logarithm then \log of the cardinality of H is also large right, the \log of the cardinality of H is basically cardinality of U times \log of m right, which is fairly large and we do not want to do that. So, therefore what we will try to do is to try to create hash families set of small in size right and will see will go into this precise in a little bit in a little more.

(Refer Slide Time: 09:30)

The slide is titled "Hash family" and contains the following content:

- We need to create small hash families H such that choosing from it gives a function with "good behaviour"
- Uniform: $\Pr_{h \in H}[h(x) = i] = \frac{1}{m}$ for all x and i
 - Not enough
- Universal: $\Pr_h[h(x) = h(y)] = \frac{1}{m}$ for all $x \neq y$

Handwritten annotations on the slide include a blue circle around the word "gives" in the first bullet point, a blue arrow pointing from "Not enough" to the universal definition, a blue circle around the universal definition, and the handwritten text "~~Uniform~~" in blue ink.

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 speaking.

So, what we need to create our small hash families such that choosing a function from it gives me a function with a good behavior. So, what is this definition of behavior? The simplest notion of a function with good behavior is one that is a uniform right, that is you could you could potentially say that the hash function that given any x the hash function maps it in to a position i with probability 1 by m right there is the probability over all h belonging to H . The event that h of x equals i equals one by m for all x and for all i , it turns out that this is not enough right. This is not enough because we could it is not very hard to construct hash function hash families that really are constant, that are mapping all elements of the universe into only one position of the of the array and still satisfies this requirement ok. So, this is not enough slightly better definition is that of universal hash function.

What is universal? The definition of universal hash function says that given any 2 elements x and y there are not equal to each other. What is the chance that h maps both of them into the same position of the array ok. If we take a minute and think about it if we were choosing h to be uniformly and random among all possible hash functions this chance is exactly 1 by m ok.

So, therefore let us just take that value right, what we are saying is that hash family capital H is called universal, if what we choose a single hash function small h from this family capital H the probability that h of x equals h of y equals 1 by m and this happens

for all pay as x and y . So, as of now it is not clear that there exists such universal family right, other than the other than the naive other than the naïve family of all possible functions, but will soon see how to construct one such family ok.

(Refer Slide Time: 11:57)

The slide is titled "Hash family" and contains the following content:

- We need to create small hash families H such that choosing from it gives a function with "good behaviour"
- Uniform: $\Pr_{h \in H}[h(x) = i] = \frac{1}{m}$ for all x and i
 - Not enough
- Universal: $\Pr_h[h(x) = h(y)] = \frac{1}{m}$ for all $x \neq y$
- Near Universal: $\Pr_h[h(x) = h(y)] \leq \frac{2}{m}$ for all $x \neq y$

The slide also features logos for IIT Gandhinagar and NPTEL ONLINE CERTIFICATION COURSES at the bottom, and a small video inset of a speaker in the bottom right corner.


Sometimes what will happen is that we would not be able to construct a families that are exactly universal and then what we go for our near universal families. So, near universal hash family is one where this equality that this probability the probability of collision is at most let us say 2 by m or some c by m and will call this a c mere universal hash family.


(Refer Slide Time: 12:29)

Page 14/14

Chaining

- When collisions happen, we store elements using a linked list from that location

 IIT Gandhinagar
Indian Institute of
Technology Gandhinagar

 NPTEL ONLINE
CERTIFICATION COURSES

11

So, why are we looking at the definition of universality. So, we looking at the definition of universality because of this particular construction. So, we have not really told you what happens when 2 elements extend by collide right. For instance look at this look at this picture that suppose this blue thing is the array that we are looking at, is the array of size m and we have positions 2 5 and 9 and position 2 we have 3 elements of the universe that have mapped in to position 2, similarly in position 9 there are 2 elements are of mapped. So, what do we do it is very simple, what we do is that we basically keep all the ids of the all the elements that are mapped into the same position using a link list that has a root at that position right.

So, because there are 3 elements set of mapped in a position 2 we keep a link list of size 3 and the root of the link list is the position 2. Similarly we keep a link list of size 2 starting at the position 9, now what happens when I get, when I get an element when I get a query I map it into the into the corresponding position of the array using the function H , then I will then I see then I look at the link list starting from that position. If there is no link list there then of course, I say know this particular query does not exist in a hash table, but if there is a link list then you go over all the elements of the link list trying to find that element.

(Refer Slide Time: 14:01)

The slide is titled "Chaining" and contains the following text:

- When collisions happen, we store elements using a linked list from that location
- $l(x)$ = length of chain at position $h(x)$
- Expected time to query $x = O(1 + E_h[l(x)])$
 - Same for insert and delete

A diagram on the right shows a bucket in a hash table containing a linked list of nodes. The bucket is labeled $h(x)$ and the nodes in the list are labeled x .

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". The page number "12" is also visible.

So, how do we get the expected time for a query? The expected time supposing suppose in l of x , so let us suppose the query is the element x right so denote by l of x the length of a chain starting at position $h(x)$. Therefore, the expected time to query the element x to say yes or no, whether this element x belongs to the hash table or not is given by this expression. So, first you have to calculate the hash table the value h of x for the element x and that is going to take out the one time.

Let us say right and after that we have to go over the entire link list l of x trying to find this element x right and therefore the time taken for that will be l of x it is linear in l of x right and because we are taking expectation with respect to the hash function h . Therefore, the expected query time for the for the element x is order 1 plus expectation over h l of x right and similarly for inserts and deletes ok. Because when you have to insert an element we have to we have to again look for the element whether the element exist and if it does not exist then you then you insert at element similarly for delete. So, now this is the quantity that we need to analyze in order to bound the expected time for query.

(Refer Slide Time: 15:28)

Page 14/14

Analyzing chaining: universal hashing

- Need to bound $E_h[l(x)]$
- For $x \neq y$, define $C_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{else} \end{cases}$
- $E_h[l(x)] = E_h[\sum_y C_{xy}] = \sum_y \Pr[h(x) = h(y)] = \frac{n}{m}$

universal

IIT Gandhinagar
Indian Institute of Technology Gandhinagar

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

15

So, the core part of that quantity that you saw is this particular expectation, expectation over h of $l(x)$. So, how to be bound this? This seems like a daunting quantity, let us try to create a bunch of very simple random variables first, for 2 elements x and y x not equal to y define the following indicator variable C_{xy} .

So, C_{xy} equals 1 if h of x equal to h of y that is h is telling you to map x and y to the same place and C_{xy} equal to 0 else. So, therefore now I can write down now I can write $l(x)$ in terms of in terms of the simple random variables C_{xy} . So, it is not hard to see that $l(x)$ is nothing but the summation over all y C_{xy} right, which is by the linearity of expectation; now we can propagate their the expectation of h inside the summation and because the C_{xy} is a really burn 0 1 random variables.

The expectation of C_{xy} is nothing but the probability that h of x equal to a equal to h of y and now we see over the definition of universality comes in. Assuming that the hash function h that I started with is exactly universal this probability is $1/m$ for all pairs x and y and because this is the sum over n values the expectation that we get is n/m right.

So, this factor n/m right we intuitively call it the load factor of the hash table. So, in some sense you have to hit this factor in your query time we cannot do a much better than this.

(Refer Slide Time: 17:19)

The slide is titled "Multiplicative hashing" and contains the following content:

- How to design small + universal hash family?
- Prime multiplicative hashing:
 - Fix a prime number $p > |U|$
 - $H = \{h_a(x) = (ax \bmod p) \bmod m, a \in \{1, \dots, p-1\}\}$
 - Choosing a hash function is same as choosing $a \in \{1, \dots, p-1\}$

The footer includes the IIT Gandhinagar logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the number "16".

So, next let us talk about how to create a universal hash family ok. We have analyzed we have defined universal hashing we have used it to analyze the query time and but we are still not know that we can create a family of universal hash functions. So, here is a very simple method to do that. So, suppose you start with a with the universe U look at the cardinality of the universe U , then choose the prime number p that is larger than this the then this cardinality.

Then define the following family right define look at all numbers a small a in the range from 1 to p minus 1 and for each such number there will be a function h_a . So, what is this function h_a this function h_a is nothing but this very simple operation first you calculate given a number x . So, we are thinking of numbers as so we are thinking of element ids as numbers in the range from 0 to cardinality of u without laws of generality you can always do that right. So, the function h of a is defined as given any such input x first you calculate the product a times x , then you take modulus of p what is that give me that gives me the remainder of a times x when divided by p .

So, first you calculate $ax \bmod p$ and then you calculate the mod m of that, basically the remainder of that when divided by m . So, now what do we have we have a number in the range from 0 to m minus 1 1 which is exactly what we desired and we use this number as the value h_a of x , which means that the place to store x in the array m . So, let us pretty clear that choosing a hash function at randomly is basically the same as choosing a

number a in the range from 1 to p minus 1 at random and this is what we will analyze. But what happens, when you create a hash function by choosing the number a from 1 to p minus 1 at random and calculating this value h_a of x do we get a universal hash family.

(Refer Slide Time: 19:42)

Page 14/14

Multiplicative hashing

- $H = \{ h_a(x) = (ax \bmod p) \bmod m, a \in \{1, \dots, p-1\} \}$
- This family satisfies $\Pr_h[h(x) = h(y)] \leq \frac{1}{m}$
- **Intuition:** $h_a(x) - h_a(y) = (a(x - y) \bmod p) \bmod m$
- There are at most $\frac{p-1}{m}$ values in $\{1, \dots, p-1\}$ that are divisible by m
- What is the probability of choosing a such that $(a(x - y) \bmod p)$ is one of these numbers?

IIT Gandhinagar
 Indian Institute of Technology Gandhinagar
 NPTEL ONLINE
 CERTIFICATION COURSES

So, let us look at, let us look at the quantity h of x equal to h of y ok. So, what we want to say that suppose h of x equals h_a of y when can that happen, remember we get to choose a , we do not get to choose x and y x and y are given to us. So, in how many ways can we choose a such that h of a x equal to h_a of y , so if that happens right then the a times x minus y mod p right must have been divisible by m , because this left hand side equals 0.

Now a times x minus y mod p is a number from 0 to p is a number from 0 to p minus 1. So, how can that be divisible by m how many how many such numbers are there that are divisible by m , it is not very hard to see these much bigger than m . So, therefore there are exactly $\frac{p-1}{m}$ values in the range 1 to p minus 1 that are divisible by m .

So, how do I choose, a in how many ways do I choose a so that I get to hit exactly one of these values and here is where we have to look at some properties of prime numbers, that is we are trying to analyze that what is the probability of choosing a such that a times x minus y mod p is 1 of these $\frac{p-1}{m}$ numbers.

(Refer Slide Time: 21:27)

A property of prime numbers

WLOG $x - y \in [1, p - 1]$ $a(x-y) \bmod p \rightarrow \frac{p-1}{m}$

Property: For every $t, z \in [1, p - 1]$ there exists unique $a \in [1, p - 1]$ such that $az \bmod p = t$ $az \equiv bz \bmod p$

By contradiction. If not, then $\exists a, b \in [1, p - 1]$ such that $(a - b)z \bmod p = 0$.

But this cannot be as p is prime.

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

So, these are class of generality say that x is x is the bigger among x and y and therefore x minus y ranges between 1 to p minus 1. In fact it is smaller, but to that loss of generality we can do that, so here is a very interesting property of prime numbers. suppose you start you take any prime any prime p right and look at 2 numbers t and z such that both t and z are in the range from 1 to p minus 1. In that case what we can say is that there exists a unique number a in the range from 1 to p minus 1, such that a times $z \bmod p$ equals t ok. So, this reminds you of the property of division right in the case of in the case let us say rational numbers right.

If I were take a if I have taken out this mod p right, if I was sort of doing a normal division instead of instead of doing a modulus this property what have been obvious to you, but it is not. So, obvious because we are doing this modulus operation; however, it is easy to show by this is the case that supposing this property is false. That means, that suppose there are 2 numbers a and b such that az equal to $bz \bmod p$.

In that case what happens; in that case what happens is that you get to say that a minus $bz \bmod p$ equal to 0 because and do convince yourself of this that this modulus operation satisfies all this linear properties. But then a and b are both less than p , z is also less than p and if the product of them is divisible by p , one of them must be divisible by p but how can that happen p is a prime number. So, therefore the product of them cannot be divisible by p the product of them cannot have mod p equal to 0 unless a equal to b .

So, therefore there exists a unique a , that satisfies $az \pmod p$ equal to t for all z and t in the range from 1 to p minus 1 right. What does this mean now we were looking for the solution for we were looking for the solution to a minus b times x , we were looking for the solution to a times x minus $y \pmod p$ to be 1 of p minus 1 by m numbers.

But what this statement means is that for each of these p minus 1 by m numbers there is exactly a unique choice of a , that is going to map x minus y to that particular number. So that means, and these are the only numbers that can cause collisions among x and y , so that means at the probability of choosing such an a is really only p . So, there is only p minus 1 by m such choices of a that cause that can cause a collision between 2 given x and y and therefore the probability of choosing such an a is nothing but p minus 1 by m divided by 1 by p minus 1 which is equal to 1 by m and that is exactly what we wanted right because that that is really the definition of universal hashing.

(Refer Slide Time: 24:59)

Page 14/14

k-wise universal

- For any distinct (x_1, \dots, x_k) and any (not necessarily distinct) (y_1, \dots, y_k) ,

$$\Pr[h(x_1) = y_1 \wedge \dots \wedge h(x_k) = y_k] = m^{-k}$$

- Needs only $O(k \log n)$ bit of storage

= = ○

IIT Gandhinagar
Indian Institute of Technology Gandhinagar

NPTEL ONLINE
CERTIFICATION COURSES

So, next we will, so while the definition of universal hashing of universal hashing of hash functions is very useful, we also made some times a little strong definitions in some sets right and this and I will show you how to construct such hash families, but it is actually failed easy right. So, here is a definition of what is known as the k wise universal also sometimes called as strongly k wise universal hash function.

What does it say what it says is that instead of taking 2 in instead of taking x and y a pair, now we take k tuple of k tuple of values from the universe x_1 to x_k and we take a k

tuple of values from the hash positions 0 to $m - 1$. So, while the k tuple here x_1 to x_k are all distinct the k tuple y_1 to y_k are not necessarily distinct, they can be few of them I mean they can be the same y_i can be the same and what we want to say is that over the choice of the hash function right.

The probability that of this particular event and this is the event over which is the and over k possible simple events the probability over h that $h(x_1) = y_1, h(x_2) = y_2, \dots, h(x_k) = y_k$ equals $1/m^k$ right. If this happens we call the function h to be as strongly k wise universal hash function. So, how do we create such a hash function? It is very simple actually, rather than just using $ax \bmod p$ we now have to use a polynomial of size k right.

We have to choose; we have to choose a 1 to a k and therefore, create a polynomial and it is not very hard to say that by the uniqueness of roots of a polynomial you get the you get this property. And in order to store this particular hash function you will only need to store the coefficients that this polynomial uses which is going to be of size $k \log n$, there are going to be k there are going to be k numbers each of them will only need out the $\log n$ bits and therefore it is enough the store order $k \log n$ bits.

(Refer Slide Time: 27:31)

Page 14/14

Summary

- Hashing
 - Simple and versatile
 - Main issue is design of good hash functions, much researched area
 - (near) universality guarantees small chain sizes
 - Other alternatives to chaining exist, e.g. open addressing, cuckoo hashing

= = O

NPTEL

IIT Gandhinagar
Indian Institute of
Technology Gandhinagar

NPTEL ONLINE
CERTIFICATION COURSES

22

So, just to summarize we looked at the very basic data structure that will use over and over in this course and that is hash table and we saw that a hash table really there is no magic. There is a there is a particular array m right where we store the elements and if

there is magic it is all there in what is known as a hash function and the design of the hash function is the most critical part of creating a hash table.

So, this is a it is going to a simple it is a simple and versatile data structure and the main issue is how to is, how to create this hash function and just Google just look up the Wikipedia page for that there is an excellent literature about and it is a much researched area of how to create hash functions that are fast, that are provable properties and so on. But we also saw is that universality of hash function or even near universality is something that guarantees small chain sizes and therefore guarantees faster query times.

We looked at we have looked at only a one particular method of handling collisions that is known as chaining, that is whenever we had a collision we used a pointer we used a we used a link list right. There are other ways of handling in chaining which are and the most conspicuous one is one is open addressing and this is this typically results in smaller spaces smaller space usage, but it is much harder to analyze. Cuckoo hashing is a techniques that is related to this and that is I mean that is a beautiful piece of work, but it is a little to advanced for our class.

So, that is it for today thank you for joining us for this lecture, the primary reference for this lecture is this lecture notes by professor Jeff Erickson algorithms and models of computation please look up this textbook. It is an online resource at this point, there are other places where hashing and hash tables have been covered in quite a bit of details the standard book of algorithms by Cormen Leiserson and Rivest is very nice chapter in it and the book on randomized algorithms by Mitzenmacher and Upfal also has a nice chapter on hashing and hash tables.

Thank you.