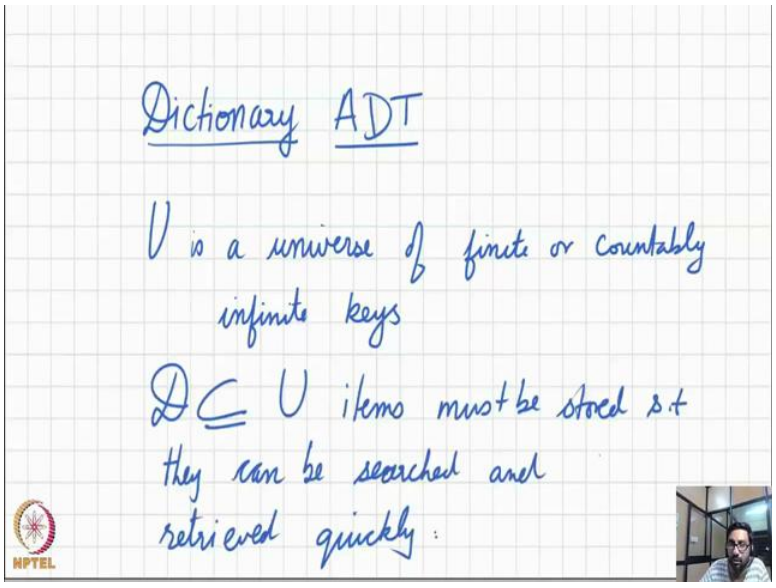
Algorithms for Big Data
Prof. John Ebenezer Augustine
Department of Computer science and Engineering
Indian Institute of Technology, Madras

Lecture – 25
Chain Hashing, SUHA, Power of Two Choices

(Refer Slide Time: 00:27)



Dictionary ADT

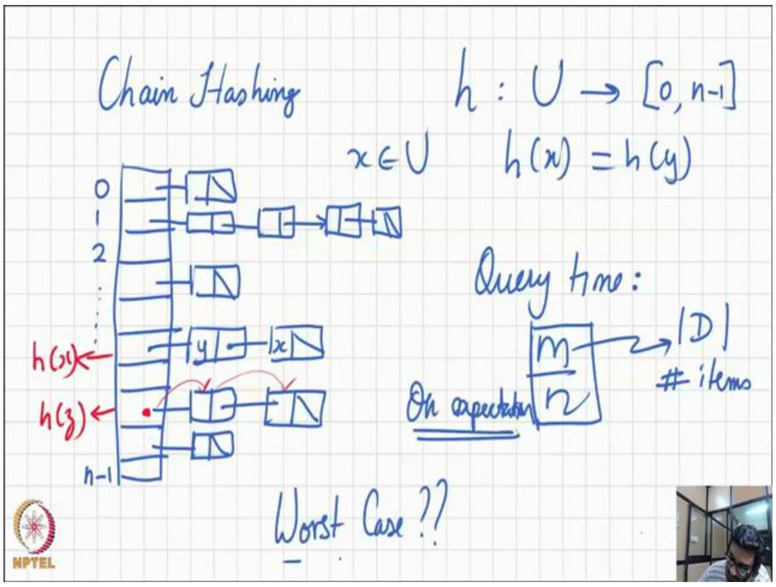
U is a universe of finite or countably infinite keys

$D \subseteq U$ items must be stored s.t they can be searched and retrieved quickly.

In this segment, we are going to look at how to use chain hashing to implement the dictionary ADT. So, let us recall what the dictionary ADT is all about that.

U be a universe of either finite or countably infinite keys, so each item that we want to store in the dictionary is identified by a unique key. Our goal in the dictionary in implementing the dictionary ADT is to store any subset d of this universe keys, and it must be stored in such a manner that you can search and retrieve those keys quickly.

(Refer Slide Time: 01:03)



You may have studied balanced binary search trees and which are very good ADT's dictionary ADT's. You may have also studied earlier on you may have studied arrays link list which also can service dictionary ADT's. What we are going to look at today is how to use hashing in particular that data structure called chain hashing to implement a dictionary ADT. In chain hashing, we have a hash table with n minus 1 cell in it. We also have a hash function, let us called that h and this hash function maps U to the range of in this is of the hash table. Whenever we need to store an item let say x belonging to U; all we do is we apply the hash function and store the element x in the index h of x, but eventually we are likely to find some other time that is also going to hash into the exact same location.

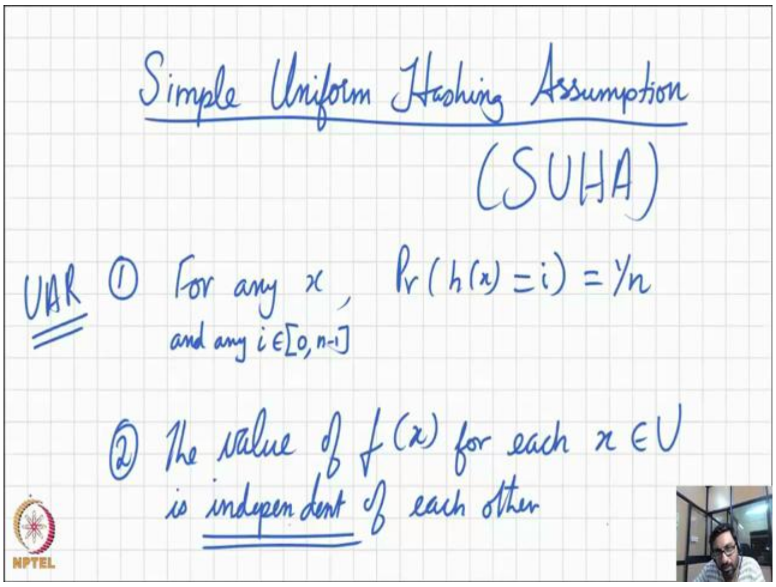
What we do at that point in time, we simply extend the list of items stored in that particular location h of x. How do we do that well we just maintain a link list. So, let us say we have two items x and y, whose hash values are the same. Let us say that x is already in the hash table and then we encounter y and y needs to be added to the hash table. So, then what do we do we insert the item y into the linked list, which is pointed to by the location h of x in the hash table.

In this manner, the items that hash into a particular location are stored as a link list. Your final picture would something like this. Based on the ways in which the hash function works, and the list of items that need to be stored in the hash table, chain hashing based hash table, you will notice that some of the link lists are going to be some more long, while others can be fairly short and even some of them can be missing. So, we are obviously interested in understanding how this in analyzing and understanding how this chain hashing works, what are it is qualities in particular we are interested in the query time.

How long does it take to search and retrieve an item. So, how does that event work? So let us say we have an item z, and we need to find out whether this item z is in the hash table. What do we do we look at h of z and that is the index h of z in the hash table and then so that index is going to point as to particular location in the hash table. And from there, we see a link list; and we follow the links in the link list to see if the particular item z is present in the hash table or not. So, we would go over here and is that z, if that is not a z then will go one-step further, and we either find z there or we do not find it in which case we just say z is not in the dictionary. And if it is there in the dictionary, well we can we are in a position to retrieve it so that is how the queries work.

And as you can imagine the queries on average are going to be m over n, where m is the size of the dictionary or simply the number of items that are stored in the dictionary; and of course, n is the size of the hash table. But unfortunately this is only on expectation and of course, are be concerned now is what about the worst case, how does a chain hashing work in the worst case. So, how do we go about analyzing that well here is where the balls and bins paradigm would be of help. Before we go about analyzing chain hashing, we have to be clear about some properties of the hash function that we used.

(Refer Slide Time: 06:24)



Simple Uniform Hashing Assumption (SUHA)

UAR ① For any x, and any $i \in [0, n-1]$ $\Pr(h(x) = i) = 1/n$

② The value of $f(x)$ for each $x \in U$ is independent of each other

In particular, we are going to assume a fairly strong property about the type of hash functions that we use. This is called the simple uniform hashing assumption. And what is it mean it has to participate, firstly, the probability with which any item x from the universe gets mapped onto a particular location i within the hash table is exactly equal to 1 over n. So, it is the hash function chooses the index i uniformly at random.

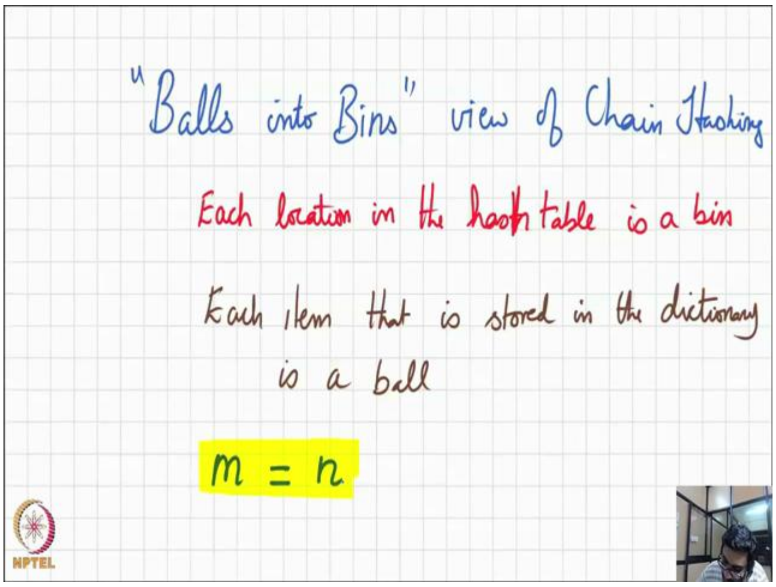
And just to clarify, this does not mean that each time x is hashed, it will be any one of the n items equally likely, no h of x will always be a particular value, but that particular value will be equally likely to be any one of the n locations. But if you were to hash the same x again it is going to fall into the same location over and over again. So, each invocation of the hash function is not going to lead to a different location, it just means that when you hash a particular item, we do not know anything about where it is going to land, it is equally likely to land in any one of the n location. So, this is uniformly at random part of the SUHA assumption.

The second part to SUHA is the independence of each hash value. So, the hash value f of x would each x must be independent of each other. In other words, the fact that f of x mapped to a particular location should have no bearing on the location to which another item y maps to.

Notice that there is a little bit of tension here, what is that tension well on the one hand each time you invoke the hash function for a particular x, we have to deterministically get back the same hash value; however, the behavior of the hash function should be random. So, for each value x from the universe, when you hash it, the probability with which it falls into any one of the bins must be equally likely and they must be independent of each other these are all classic expected things that we see with uniformly at random choices.

How do we combine these two well there is a lot of literature on that and that is beyond the scope of our current lecture. So, we are not going to delve into that for say, but what we are going to instead do for our purposes is just make this assumption that such a hash function exist and that is what we are using. And our entire analysis is going to be built on this simple uniform hashing assumption.

(Refer Slide Time: 09:54)



"Balls into Bins" view of Chain Hashing

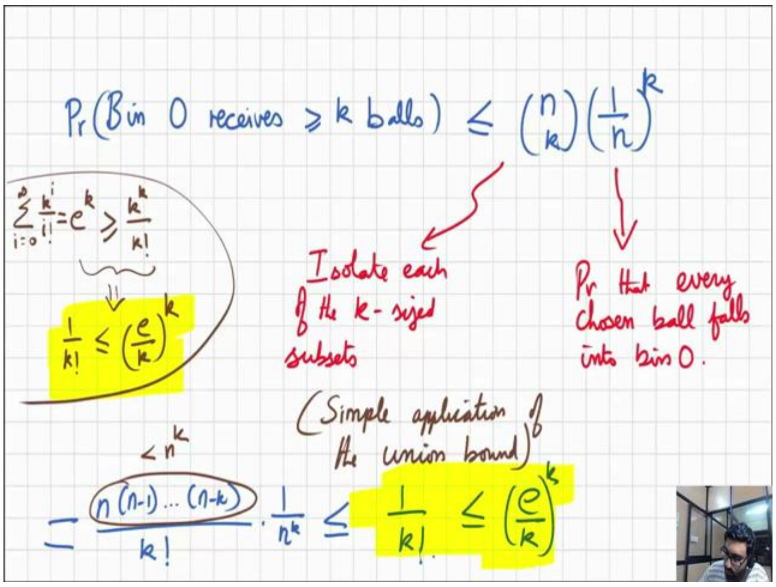
Each location in the hash table is a bin

Each item that is stored in the dictionary is a ball

$m = n$

And as you can imagine we are going to use balls and bins to view this chain hashing problem. We are going to view each location of the hash table, each location in the hash table as a bin; and each item that is stored in the dictionary is a ball. And this ball has thrown uniformly at random into one of the bins, and that is by virtue of our assumption that the hash function maps an item from the universe into one of the locations within the hash table chosen uniformly at random. And to simplify the analysis, we are going to assume that the hash table size that is n is chosen to be exactly equal to the number of items that we want to store in the dictionary; in other words, m equal n.

(Refer Slide Time: 11:03)



$$\Pr(\text{Bin } 0 \text{ receives} \geq k \text{ balls}) \leq \binom{n}{k}\left(\frac{1}{n}\right)^k$$

$$\sum_{i=0}^{\infty} \frac{k^i}{i!} = e^k \geq \frac{k^k}{k!}$$

$$\Downarrow$$

$$\frac{1}{k!} \leq \left(\frac{e}{k}\right)^k$$

Isolate each of the k-sized subsets

Pr that every chosen ball falls into bin 0.

(Simple application of the union bound)

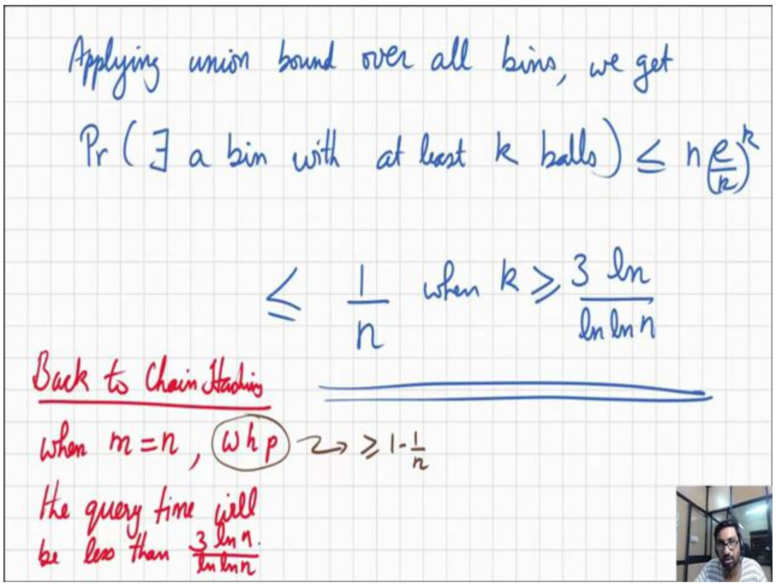
$$= \frac{\overset{< n^k}{n(n-1)\dots(n-k)}}{k!} \cdot \frac{1}{n^k} \leq \frac{1}{k!} \leq \left(\frac{e}{k}\right)^k$$

Under these assumptions, we are interested first of all in the in the event that bin numbers 0 in other words the very first hash location gets care more items. The question is what is the probability of this event and that is upper bounded by n choose k times 1 over n raise to the k. So, the n choose k will isolates some k items from the list of items that we are storing in the in the dictionary and there are n choose k ways of doing it. And once such a subset is chosen each ball in that subset should fall into bin 0 and that is 1 over n raise to the power k.

Notice that, we are potentially over counting the probability here because what we are interested in is the events that bin 0 receives more than or greater than equal to k balls. What we have on the right hand side is we are isolating each of the k subsets and applying a union bound over all those k sized subsets. And this n choose k times 1 over n raise to the k can be expanded, in fact, it is in quality can be expanded into n times n minus 1 times and so on up to n minus k divided by k factorial times 1 over n to the k.

Notice that the numerator here is at most n to the k. So, this whole quantity is upper bounded by 1 over k factorial; and this in turn is upper bounded by e over k the whole raise to the power k. How do we get this we know that e to the k can be expanded using the Taylor series. And we will get the summation over from i equal to 0 to infinity k raise to the i divided by i factorial, but in that summation let us isolate just one term, so that is the term k raise to the k divided by k factorials that is when i will equal k. So, obviously, if you just isolate that one term the relationship here is going to greater than or equal to and of course, this will imply that 1 over k factorial is at most e over k raise to the power k and this inequality that we get over here is exactly the inequality that we use over here.

(Refer Slide Time: 14:06)



Applying union bound over all bins, we get

$Pr(\exists$ a bin with at least k balls$) \leq n\left(\frac{e}{k}\right)^k$

$\leq \frac{1}{n}$ when $k \geq \frac{3 \ln}{\ln \ln n}$

Back to Chain Hashing

when $m = n$, $\bigcirc{whp} \rightsquigarrow \geq 1 - \frac{1}{n}$

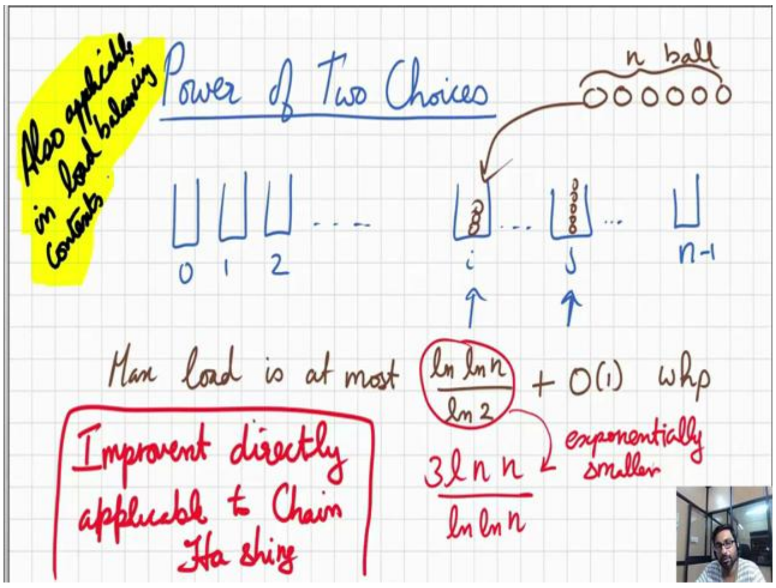
the query time will be less than $\frac{3 \ln n}{\ln \ln n}$.

So far we only focused on bin 0, but if we want to include all the bins, we simply apply a union bound over all the bins, and so we get the probability that there exist a bin with at least k balls is at most n times e over k whole raise to the power k. And from here, it is a simple exercise and encourage you to work it out is a simple exercise to show that this quantity is at most 1 over n, when k is at least 3 ln n over ln ln n and this is in fact one of the most classic balls and bins resolves. If you throw n balls each into a bin that shows an uniformly at random and independent of each other; if you throw these n balls into n bins, the height or the number of balls that fall into the bin with the most number of balls is at most 3 ln n over ln ln n with probability at least 1 minus one over n.

In other words, if the height of the max loaded bin will exceed 3 ln n over ln ln n with probability that is upper bounded by 1 over n, which is the way we have stated it over here. So, applying this result back to the chain hashing context, we can conclude the following, when m equal to n that is when the size of the hash table is designed to equals the size of the dictionary that we want to store.

Then with high probability that is probability of the form 1 minus 1 over n the query time will be less than 3 ln n over ln ln n. So, this is significantly less than what you might normally imagine because the worst case query time could be as high as n when all items hash into the same location, but when we apply the simple uniform hashing assumption, we get a much better bound and this is much more realistic as well.

(Refer Slide Time: 16:48)



Power of Two Choices
Also applicable in load balancing contexts
n ball
0 1 2 . . . i . . . j . . . n-1
Max load is at most $\frac{\ln \ln n}{\ln 2} + O(1)$ whp
$\frac{3 \ln n}{\ln \ln n}$
exponentially smaller
Improvement directly applicable to Chain Hashing

As it turns out there is further improvement that we can achieve as well but this will require us to use two hash functions instead of one hash function. So, this is a very classic well known result called the power of two choices. Let me first state this result from the perspective of balls and bins. So, we have n bins then the index from 0 to n minus 1 and we have n balls. Each ball chooses two bins both uniformly at random and independent of each other and independent of previous choices. And so let us say without loss of generality bins i and j are chosen. And then we check the number of balls that are already present in i and j, because this process is been happening several balls have already been placed.

So, now we checked to see the load on bins i and j. And we find out that one of the bins has fewer balls, and so we place our ball into the bin that has the fewer balls. And this is simply the power of two choices each, for each ball we simply probe two bins, and out of those two bins we find the bin that has the fewer number of balls and place the current ball into that bin. And quite surprisingly this small variation of the classic balls and bins it gives us a much better max load meaning the a much better bound on the number of balls that is going to be found in the bin with the most number of balls.

In fact, the max load is going to be at most ln ln n over ln 2 plus O of 1. Now that has to be contrasted with the previous bound that we got which was 3 ln n over ln ln. And so notice that this bound is exponentially smaller than the previous bound that we got. And of course, it is easy to see that this improvement can directly be applied in the context of chain hashing provided we use two hash functions. And not surprisingly this improved result is also quite applicable in the context of load balancing, cloud computing, and data center optimization and so on and so forth this one.