NPTEL
NPTEL ONLINE CERTIFICATION COURSE

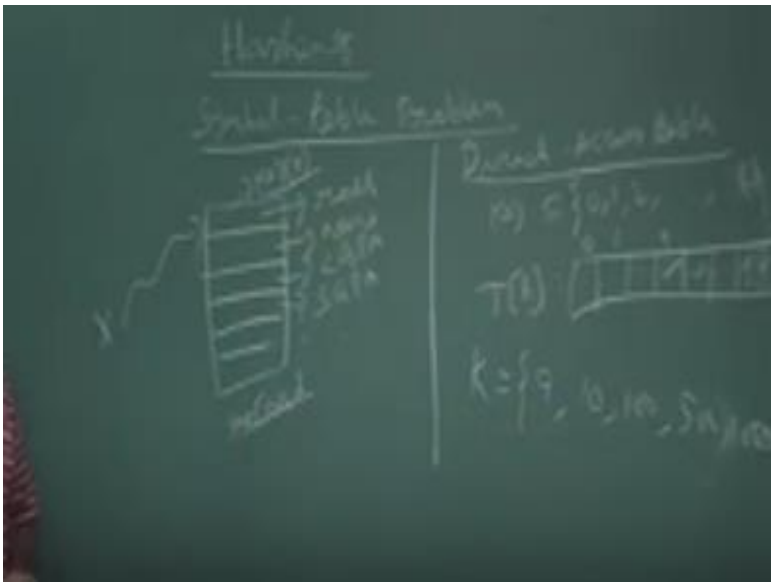
Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhopadhyay
Department of Mathematics
IIT Kharagpur

Lecture 09: Hashing

Okay, so we will talk about hashing, so the problem is which is called symbol table problem.

(Refer Slide Time: 00:38)



So this is a very common problem in the compiler design, so basically we have some records, we have AM records, now the question is how we store the record in the table. So records are typically say, may be for a student there are many fills, student roll number, student name, student marks, CJPA, semester wise mark, SGPA, Address and so on.

So let X be the pointer corresponding to this record, this is one record. So now, so this all of this filled up the record we call as a key(X) which is unit to that record. So we want to store many such they got, we know it based on the key hallow of the record. So this is the problem, how we can maintain a table to have this. So one solution is direct access table so it is basically an array suppose we know that our records are if value of the records are coming from this end.

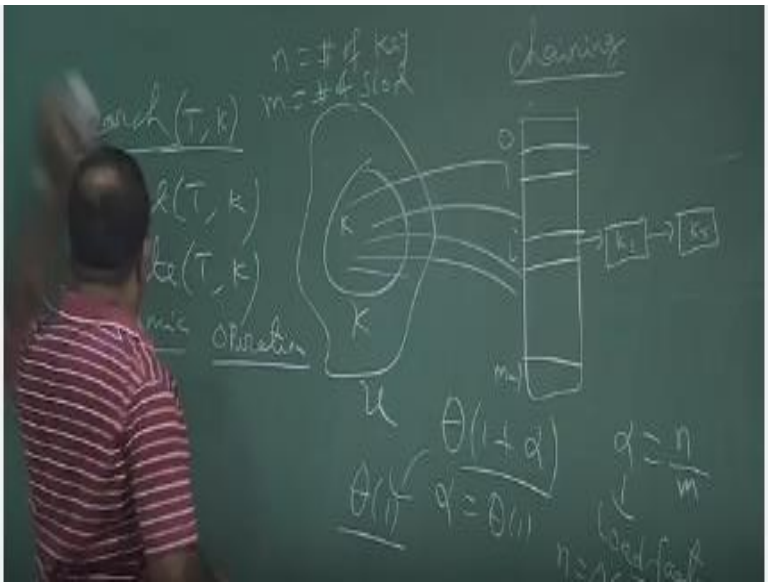
Suppose this our subset L some numbers this key value can take up to this. So we maintain a table of basically a array of size l, we can take this T[L] so this is basically a array 01 or may be this is l-1, so l-1 is the maximum hallow of the record, maximum hallow of the key of the record can take.

So now suppose we have a record say where key hallow is, this is our records is 9, 10, 100, 500 like this. So in that case we need to maintain a array of 500 and we hope this hallow suppose this is … line, so we put a 1 over here or we can put the pointer of that record and 10, then we have 100, then we have 500 somewhere here, so this array.

So just this is a bit vector if that key is present or we will put it on or we put it up or we can store the pointer there to access the whole record. So this is what is called direct access table and remaining hallows are 0, now what is the problem with this, problem with this suppose our key is just 4 and the key hallow is huge I mean may be it is till the acts or something.

Then we have to maintain the array up to till that, so that is the problem. Although we are just have, dealing with the five keys or less number of record. So to avoid this problem we define a function which is called hash function.

(Refer Slide Time: 04:21)



So in the hash function we fix the table size suppose we fix the table size from 0 to m, 0 to m-1 there are m slot in the table and suppose u is the set of all possible keys all possible keys.

This is a universe of key, key universe, universe of key. Then the hash function is a function from this universe to this slot, so it is a function from, it is taking a key and it is mapping to a slot. So suppose this is our universe of the key and among this, this I all set of key of our interest and these are the key suppose there are say keys K1, K2 like this and, so we apply hash function on it, suppose H of this is, so H of K1 is 1 like this to like this.

So similarly say H of K2 could be here some say i value. So this is H of K2 and different, different keys are mapping to this slot, this table, into the table. Now the problem is suppose we have it K5 for which a's value is also I so this $h(k_2)$ and also $h(k_5)$ which is also mapping to the same slot so that means $h(k_2) = h(k_5)$ which is basically I then this situation is called collusion then this situation is called collusion now the question is how we can handle the collusion because this is just a slot so it cannot store 2 records so we cannot have 2 record in single slot.

So this is the problem of hashing when we have a collusion when two keys are mapping to a two or more than 2 keys or mapping to a same slot then we have problem so how to handle this problem one to way handling this problem is by chaining method the collusion so suppose in the i^{th} slot we this the i^{th} slot so k to k_5 are colliding so what we do we will put a chain over here link this kind of thing so k_2, k_5.

So this the way we handle the collusion so whenever a slot is having more than 2 element more than 1 element more than 1 element so we will put them into the list link list so this called basically chain okay so now we analyses this chaining method. Okay so this is our table 0,1 up to m-1 and this is our set of k's of internal this is universe of the key and suppose n is the number of keys which are mapped here k_1, k_2, k_n and aim here is the number of slot okay.

Now suppose we want we know that how our hash function is good in the sense that it is uniformly distributing the keys over this slot that means if found any key it is equally lackey to put any of this slot with equal probability so that is the uniformity of key distribution by the hash function, so we want the hash functions would have this property that it should distribute if would any key k and then h(k) will be equally lackey to be in one of this slot so that is called uniformity .

I mean that is usually good hash we will see why it is good so if we assume that, that our keys are very much equally lackey sorted into this slot then while we search how much time we are taking to search a key so basically we need to search a key so we need to search whether a key is in the table or not this is one operation we need to do another operation we need to do the insert we need to insert a key.

So insert a simple we will take the key and we will go to that slot and if it is empty we will but it there or if is already chain we will put it into the chain like this delete a key so we for deletion we need search that and then we delete from that so these two is called dynamic operation because this is insertion and deletion of the record this is dynamic operation so our symbol table should able to handle this, so we want to maintain a, we want to have a data structure for our

record set of records, we want to maintain a data structure for our records so that we can perform this dynamic operation as well we want to perform this search operation.

We want to search the key whether this is in or record or not okay, so now the question is how we can search a key a key so suppose which one what we do we apply the hash function on it so we will go the particular slot and if the slot is empty then the key is not there otherwise if the slot is having a chain we have to search for so this is basically 1+ alpha so what is alpha, alpha is basically n/n.

N is the number of keys n is the number slot so if we assume the uniform distribution of this keys on to the slot then this is usually the expected number of keys in each slot so that that is expected size of the chain and this what is called load factor .

This is uniform distributed keys are uniformly distributed about the table, okay. So now we will come to how we can construct with some has function, so some constants on hash function we will see.

(Refer Slide Time: 13:28)

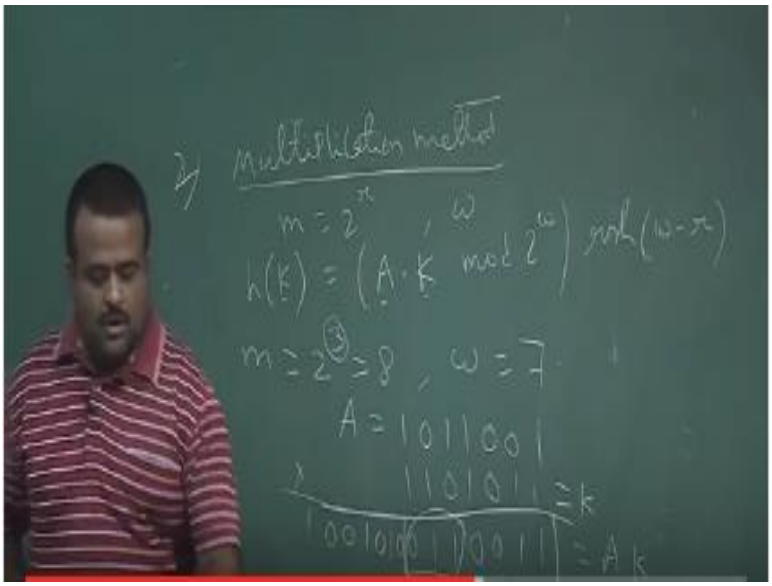
Okay so some example of hash function, so first one is division method, okay. So suppose we have our slots are m slots 0, 1, 2 up to m-1 and suppose our keys are integer so each of k is simply k mo n, okay.

So this is the division method able to in any integer if you do the mod operation so this is basically we divide that integer by a and this is the remainder, so reminder will be lying from 0 to n-1 so we will go to that slot and we will put it there, so now the question is how good this hash function? Well this will depend on the choice of n suppose our k is say 8 bit 1, 2, 3, 4, 5, 6, 7, 8, okay.

Now ours suppose our aim is 2^3 or yeah so 2^3, so the m then this mod 2^3 is basically what? This mod 2^3 is basically this value this one, so whatever value are here all the if we choose different values for this base then all the values are colliding here, so there is a huge number of collision it is not caring of this most significant bits, so the choice are being in this form is very bad 2^r is not good choice.

So we will try to avoid to choose n to be 2^r or 10 to the for some power, so usually for this method we chose n to be a prime number which is not close to m is a prime number which is not close to 2^r or 10^r or somewhat so that is the otherwise it will give us huge collisions so it will not be the uniformly distributing the keys or what the slots, okay. The second method is the multiplication method. So here instead of division we take the multiplication.

(Refer Slide Time: 16:44)



So this is the second example of hash function, this is called multiplication method, okay. So here suppose here we are allowed to choose $m = 2^r$ and suppose our computers are w bit code so we choose h or k, k is w bit is basically we take a constant a is a w bit constant, k is w bit then mod 2^w so then if we multiply 2 w bit it will be 2w bit.

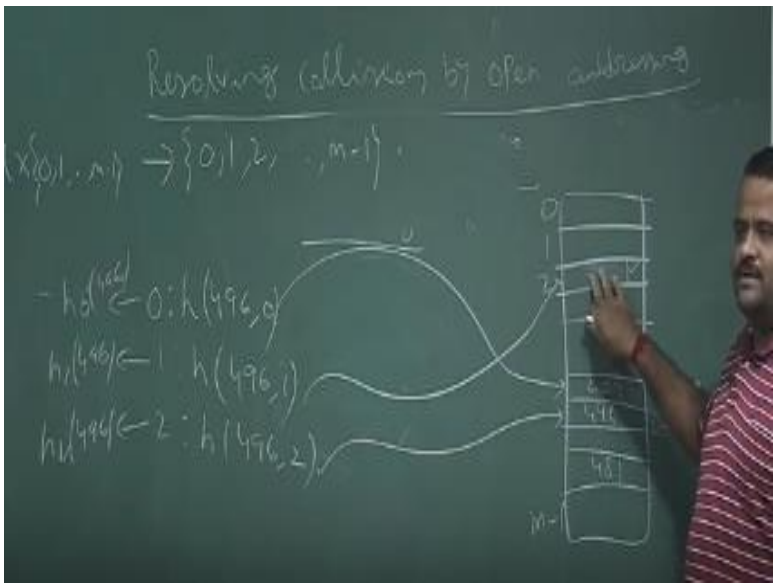
Then if we take this mod it will just give us t7he w bit then we will do a right shaped of this minus r-bit, okay. So this will give us a so we will take an example suppose we choose $m = 2^3$ which is basically 8 and we choose w = 7, okay now suppose we choose a constant a to be say 1, 0 so 7 bit 11001 sop this our constant A. Now suppose we choose a k which we want to apply the hash function 1101011.

Suppose this is the k and we want to apply each of k so if we have to apply you have to multiply this , this the binary multiplication and if we use the calculator we will be getting like this, so this is 10010100 you have to check the bits 110011 1, 2, 3, 4, 5, 6,7 so this is the multiplication of A and A. k and then you have to take the mod 2^w which is basically leave out this part and then after that we have to take the left shift of w-r bit.

So w is 7 r is 3 this is r so 4 times so if we just do the 4 times shifting then this is our h(k), okay. So this is a, this is availably better than the division method but this involve some multiplication so which may take some time, okay. So now so this there are you can construct summation or multiplication on and some other way to construct some hash function, o0kay. So now we will talk about open addressing.

By which we can avert the e can handle the collies, one way of handle the collision is chaining but for the chaining we need to have a laying is outside the table, but suppose we do not have any memory available outside the table so everything we have to do in the table in the array. So how we can handle that we are not allow to have any extra laying the outside the table.

(Refer Slide Time: 20:32)



So that is why the, what is call open address, so resolving collision by open addressing method. Okay, so, so here no extra storage, so everything you have to do inside the table so no extra storage for doing the chaining so that means no chaining, so here what we do we will, we will so suppose we have some keys we want to store it here, so we will have a proof we will try so first we will apply the hash function and if it is empty, if it is free then we will put it there. But if it is

not free then we will we are second at time and we will apply second say next prove like this, so this way we will proceed.

So basically we will do like this, so suppose we want to insert this K=496 and we have already some numbers over here is a so 586, 133, 204, 481. Okay, so the 0^{th} proof what we do we apply this 496 this is the 0^{th} proof and suppose it is hidden here, okay. so now there it is a collision and we are not allow to have a chain so what we do, we will have another attain, so we have a sequence of hash function, so we will choose the second hash function that is the next proof and suppose it is giving as this value, which is already occupied the we have to go for a say third proof, next proof.

I mean, so we have a sequence of hash function you have familiar of hash function and we keep on trying until, until and unless we get a this free space. So this is the next proof, so suppose this is giving as some free sort and we are putting it 196 here and suppose this is as i^{th} slot, okay. And so here our hash function is basically so this is basically we are handling with a sequence of hash function like this, so it is basically u x this is proof sequence so how many time almost this times to 0,1 up to n-1. So basically we can denote this by h0(496), h1(496), h2(496) so we have, we have sequence of hash function so we will try with the first hash function the original hash function if there is a collision we will tie with the next, so this sequence is very important, okay.

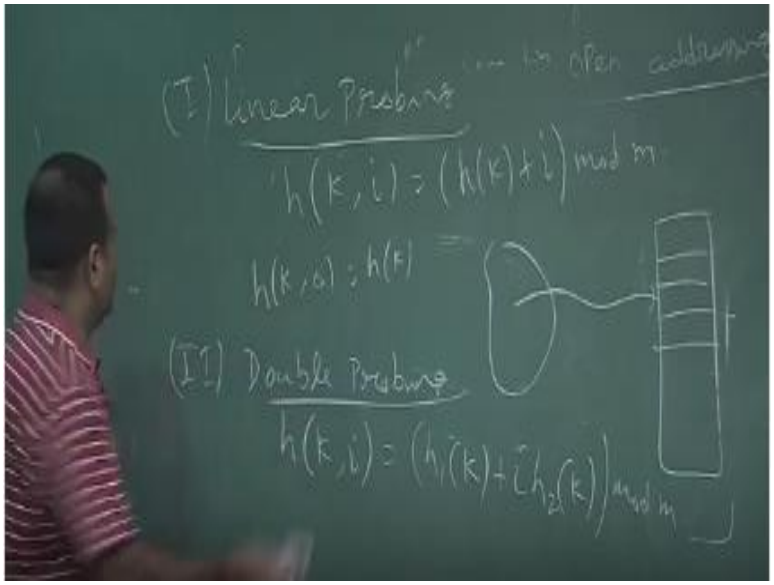
Now how we can search we have to follow the same sequence while searching and then we get the key after if the, if we achieve this by the this number 2 proof then we will get the key after this number 2 proofs, okay. But this method is having a difficult the deletion now what able to our method should handle the dynamic or personal like inside and deletion, inside is this way but deletion is problem because if we delete some number over here, suppose if you delete this, this key, okay.

So then when we try to search 496 after deleting this then what we do we just, we just go and search we will just apply 0^{th} proof between it will hit here, we will check whether this is matching or not, this will not matching then second proof will hit here but we see this is deleted I mean we do not know whether this is deleted or, or nobody was there, so we will say no this is

not in the table. But which is wrong, so we have to have a tick over here that somebody was here but it got deleted.

So then we will go for the next proof and we got the key okay so now we will take some example of the going so first one is linear probing.

(Refer Slide Time: 26:30)



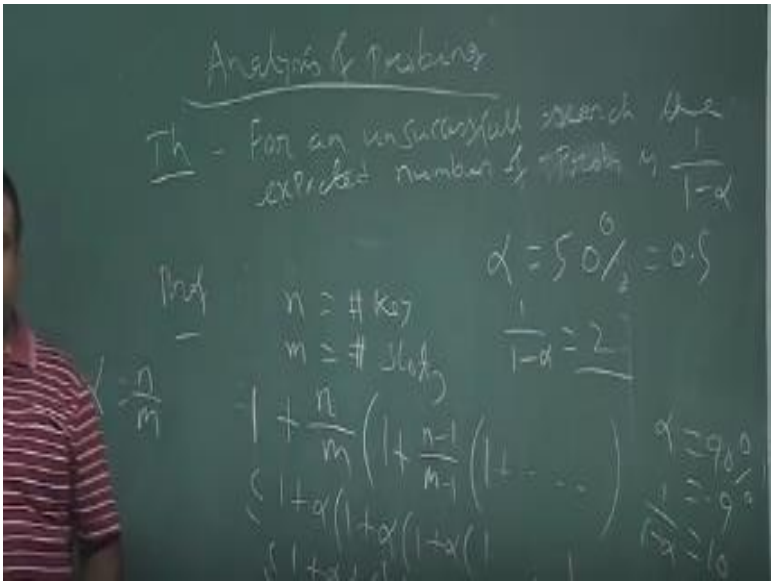
So in linear probing it is basically we compute h (k + i) mod modern this is linear probing that means if this is our table so we apply h of this fast so this is 0 if i=0 then it is basically. H(k) modern obviously now if this is not if this is not empty then we have to go for i+1 next slot like this okay these why we will keep on search until we get a empty slot but problem with these method is it will sort of create a cluster in the whenever T is there it will sort of create a cluster below that so it will not be the uniform distribution of the key over the slot so to avoid that will have a what is call double proof double probing.

So this is basically h(k , i) basically we have to has function modern so for the 0 proof it is the h well of k that could be or original function because this for this 0 otherwise if it is not getting a free then we have to computes to and you have compute h1(k) + h2(k) modern if it is still not

free then you have to compute so we have already computed h1 : then you have to keep on try for a these slot by adding this with the h2.

Okay so the this two are the example for probing now we will analyze the probing strategy or probing method analysis of probing.

(Refer Slide Time: 28:39)



So this analysis you are climbing that for a unsuccessful search we need the number of expected number of probs is required that there is a theorem, theorem is telling that for an unsuccessful search successful search the expected number of probs is 1/α so how to prove that okay suppose there are n keys and which are occupied into n tables.

n slot our table size is n okay so one prob is required and for this is the expected number of one prob is required now when you go for the second prob if there is a collision now when there will be a collision there are n keys occupied in n slot so what is the probably of collision it is basically n/m so this is the probability of it is hitting a slot where the number of key because there are impossibilities m slot but we want to hit a slot which is not mp so there are n wise we can choose that similarly after that.

Next prob is required with probability n – 1/n – 1 next prob like this so this is basically less than = 1 + α x 1+α x 1 +α like this so here α is the low factor so this is basically giving us 1 + α + α2 so this basically 1/1 – α so expected number of prob is less than = 1/1 – α so if α is same 50% then that is our table is α 50% is our table is full half way then this is basically α is .05. 5 sorry and then the expected number of prob for a answer is full search is two.

Okay now if α is 90% of the, that it is .9 then 1/1 – α is 10 so 10 prob is required for a unsuccessful search, thank you.