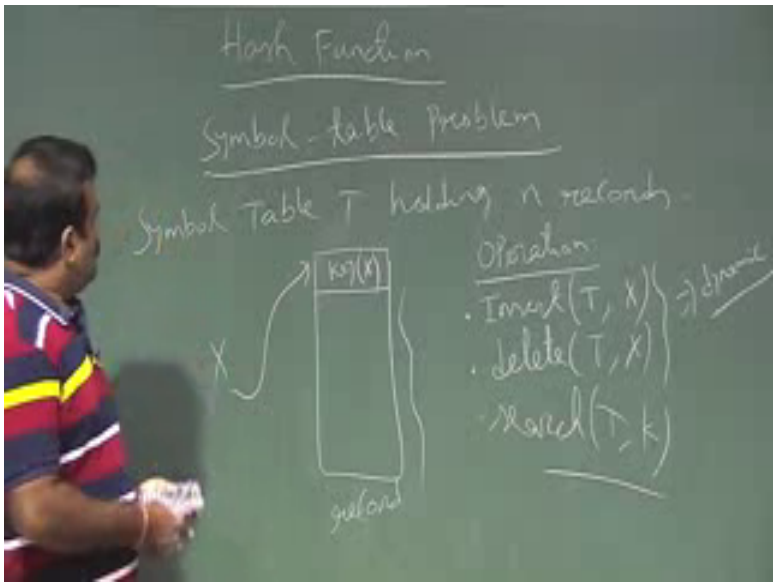
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture – 21
Hash Function

So we will start the hash function will talk about hashing. So, the problem comes for the what is called symbol table problem.

(Refer Slide Time: 00:29)



So, what is the symbol table problem? Basically we need to store. So, T is the table which is holding. So, symbol table T basically holding n records. So, we need to store the n records. So, what are the record basically? So, record is having few fields and which is say x is the pointer pointing to this record, and among this field there is one field which is referred as key of x which is basically unique and remaining are some other data field may be satellite data or something called so, but one field is unique identification of this record. So, this is one record, this is the record. So, may be this is a student record. So, student has roll number name age c g p a s g p a address.

So, these are all field, but we need to find out one field which would be use for unique identification of the student may be student roll number or student pan card number. So, that is the key of x. So, the problem is to maintain n records. So, we need to store n records in a table. So, that is the problem. So, that is called symbol table problem. So,

now, we need to have a data structure to store to maintain this record, such that we should able to perform few operations.

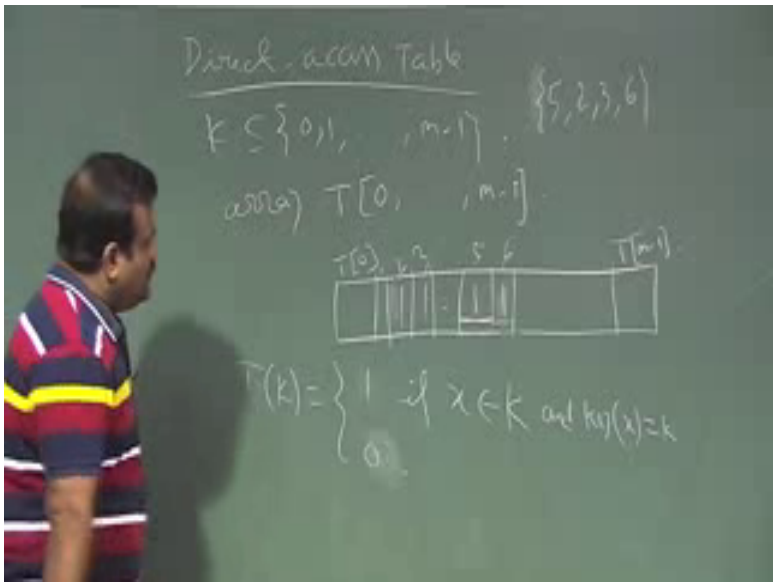
So, what are the operations on T? So, basically operations are basically three basics operation insertion. So, we should able to insert a record in the table and we should able to delete a record from the table. So, this two is the dynamics, this two will make the record dynamic. So, basically we have dynamic state of records. So, we have already seen one dynamic set where in the priority queue that heap when you talk about heap data structure to which is basically the implementation of the priority queue.

So, there we are having a set a which is a dynamic set. So, any point anytime anybody can join and anytime anybody can leave. So, we should able to have that query maximum of that set, if it is max we take max heap or minimum of that set or we should able to decrease something. So, this way, this is basically the make this set dynamic. So, we have a record set of we need to maintain set of n records, and this set is dynamic in a sense that any point of time anybody can join into that table or anybody can delete and we should able to search a record that is very important query.

So, we should able to search a record. So, key value is k. So, given the key value k we should able to find whether that record is there or not. So, given a student id we should able to find out the student record is in our data base or not. So, student record is there in our table or not. So, this is the problem. So, this is the problem called symbol table problem. So, we need to have a data structure for this. So, we need to store the n record in such a way that we should able to perform this operations.

So, we should able to insert a new record, we should able to delete a record the this two operation makes this say dynamic and we should able to search and this should be in a faster way. So, we need to have a data structure for this. So, let us just think about what is the data structure we can use for this. So, let us start with the very simple data structure, but very powerful array simple array. So, this is called direct access table,. So, here we are assuming the keys are coming from some set of values.

(Refer Slide Time: 05:06)



So, we are assuming that the keys are coming from this set 0 1 up to m minus 1. So, this is the. So, maximum value of the key is basically m minus 1. So, this assumption is required. So, then basically we set an array it is simple array m dimensional array like this. So, we basically have an array. So, this is T 0, T n minus 1 and this is direct access means. So, now this array, if a record is there in the table, then we put that value on other way sort.

So, T of k is basically the record we got if x belongs to K and key of x is basically small k. So, and otherwise it is nil. So, basically or we can have a 0 1, 0 1 bit and, but. So, we put this 1 if the corresponding say this is 5, if say suppose this is some at some time our T our record set is this say 5, 2, 3, 6 suppose this is our record set. So, 5 then 3 must be here. So, 3 these are 6.

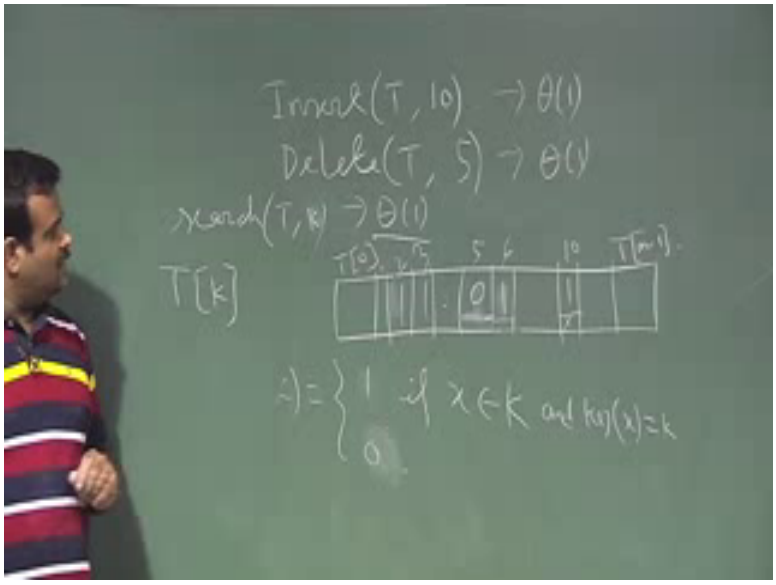
So, these are all one then say this is 2 2 2 1 0. So, these are all 1. So, this is just a bit vector 0 1. So, if that particular record is there key value then we put it one otherwise we put it zero. So, that is it very simple data structure this is called direct access table. So, 0 1 bit we can just maintain this array by 0 one bit if the record is, but somehow we need to have some information about the record where from we can get. So, we can store some pointer of that record. So, this will indicate the record is there are not.

If the record is there we can have some pointer to access that record, but anyway those are implementation issue, but we are just concerned about the presence or absence of the

record. So, if the key value is the key value 5 is there so; that means, we have a record present whose key value is 5 so; that means, this is one and then we can maintain a field over here, which will give us the exact address or the pointer where we will get the face the record I mean the whole record whole data ok.

So, this is the idea, this is the idea of the direct access table now if we use this simple data structure, now what is the time complexity for those operation how to insert a element, suppose we want to insert say 10, we want to insert a record whose key value is 10.

(Refer Slide Time: 08:31)



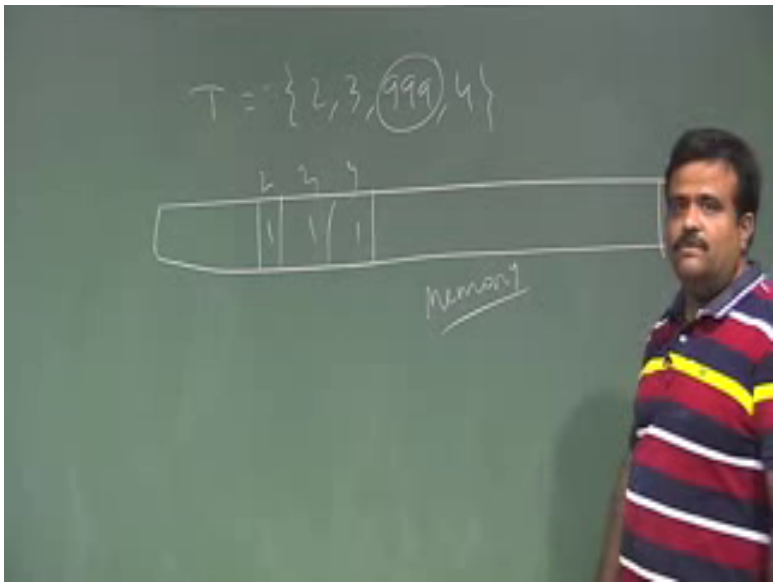
So, what will do? We go to this 10 and we put it one and somehow if we are maintaining that we will put a x over here, that will corresponding to this record. So, that is basically linear time operation I mean not linear constant time operation.

We are going to that particular field and we are putting the switch on that is it. And deletion is also similar way if we want to delete say 5. So, what we are doing? We are going there we are putting 0 we are making this empty. So, deletion is also very constant time and searching a record suppose we want to search a record whose key value is say 3. So, what we do? We go to that position go to that array position T of. So, search k say T of k now if T of k is o1.

Then we got the record and we somehow if we have the information about the record physical record we go to that position and we get the record. So, T of k depending on the value of T of k, if it is 0 then the record is not there. So, it is simply say no record is not there or if the T of k is one we got the record. So, this is the search. So, search will also take theta 1 time. So, this is the this is all are constant time operation this operation can be done in constant time.

So, this is very simple data structure, but very powerful this is just a 0 1 bit vector, but these has a problem this data structure has a problem in the sense that the memory problem; because suppose say we know there will be number of records will be less.

(Refer Slide Time: 10:36)



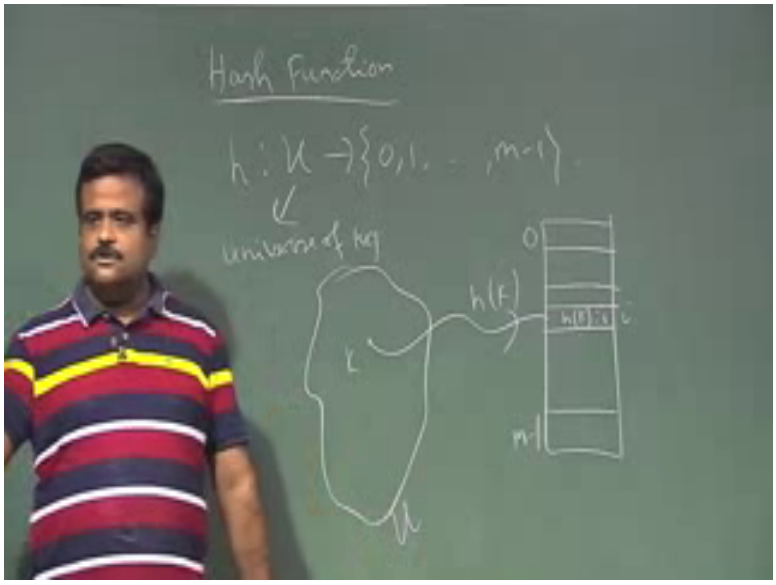
So, may be maximum we can have 6 7 record say or something more, but the size of the record is more. Suppose we have at some point of time we have this size, so 2 3 9 9 9 and 4.

So, this is the at some point of time this is the situation snapshot of the set dynamic set. Now we are allowing the record size to be this. So, for that we need to maintain the array of this many long size although we have only. So, 9 9 9 may be longer than that, but we have only using few bits. So, that is the memory problem. Memory problem because if the value is more if the value of the record key value of the record is more although the number of records is less then it is the wastage of the memory.

So, this is the major drawback of this simple array data structure, although this is very powerful this is just a 0 1 bit switch on off simple very simple data structure, but this is a problem with the value we are allowing for this key value. If the key value is we are allowing more then we need to maintain a this is statically static allocation this array. So, we need to have this data structure, we need to have the array size up to this the maximum value we are allowing for this key value.

So, this is the drawback although our number of record is less. So, to avoid this drawback what is the solution is to have a function which is called hash function.

(Refer Slide Time: 12:18)

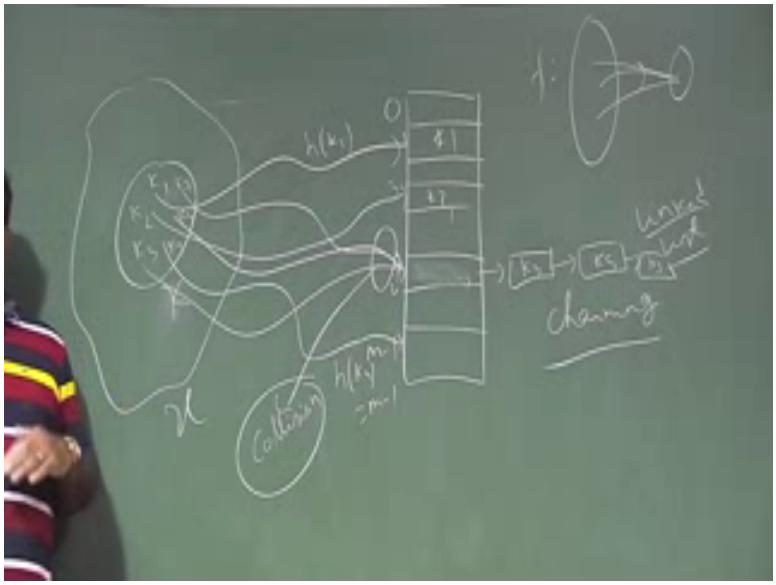


So, basically if hash function is a function from u to this set. So, suppose we have. So, u is the set of. So, this is the universe of the key, universe of key and this is the slots I mean this is the table size. So, we have say table of. So, there are m slots.

So, we have a table or this is a simple array and this is the universe of the key. So, set of all possible keys this is u. Now hash function is a mapping from u to this set. So, if you take any key of form here and if we apply h of this k it will reach towards a slot over here say i, i th slot. So, this basically h of k is i. So, any such function is called hash function. So, it is basically taking a key and it is giving us a slot basically, it is giving us a value from 0 to suppose we have given.

So, this is we have given we have given we are allow to have table size up to m. So, 0 to m minus 1. So, our hash function will be we take a key from here and it will map to a particular it will map to a slot from 0 to m minus 1. So, any such function is called hash function. Now suppose we have a hash function then how we can maintain a record. So, so let us draw this ok.

(Refer Slide Time: 14:30)



So, this is a function from key to this say 0 to m minus 1 and this is the set of all possible keys and this is the set of keys of our interest.

So, this the key set we have so far. So, here we have some k 1, k 2 some keys are there, k 3, k 4 like this. So, now, we have to maintain that stable. So, what we do we apply the hash function on k 1 suppose it is mapping here. So, h of k 1 is basically one say this is 1. Now k 2 say h of k 2 is basically say three something like that. So, this way now h of k k 3 is basically say some slot here i th slot say i. So, this is basically h of k 3 is i now suppose h of k 4 is basically say some slots here.

So, h of k 4 is say m minus 1 something like that. Now suppose we have a k 5 whose has value is say this. So, suppose h of k 5 is also mapping to the same slot i. So, then we have a problem and this is what is called as collision, this situation is referred as collision. Collision means suppose we have a 2 key which are going to map into a single slot and that is quite possible because if the slot is if this set is small and if this set is bigger.

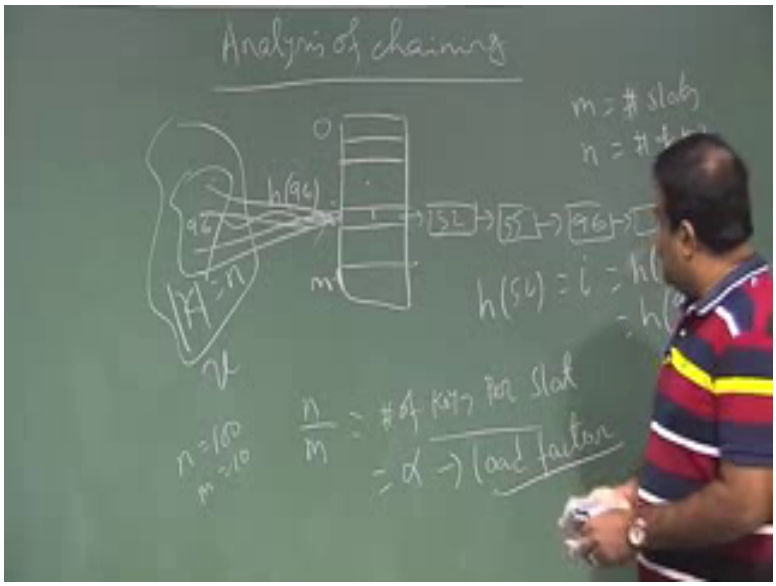
So, if we have a function say h is a function if we have a function from a bigger set to smaller set. So, then there has to be collision. So, collision is quite natural in the situation of the hash function because usually this set is smaller set. So, basically hash function is basically as a compression function. So, we have a big length input. So, we convert into small than output. So, that is the compression function. So, since this is a smaller size this co domain, this is the domain this is the co domain this smaller. So, they are has to be collision ok.

But now the question is how we can handle this collision. So, collision will be there. So, how to handle this collision because here h, so h 3, the k 1 is here k 2 is here k 4, sorry k 2. So, k 3 and k 5 both is collating to the i th slot, but they cannot sit in the single slot here. So, then what is the solution. So, we have to do some sort of chaining over here. So, because there are because this is a position for only one guy this is a room for one.

So, we can have a something what is called chaining or linked list. So, k 3 then we can have k 5. So, this is called chaining method to handle the collision. So, this is basically linked list, if a slot is containing more number of keys then we will put that outside the table will put a chain linked list, this is basically linked list. We will put a linked list outside the table. So, this is the way we just handle the collision ok.

So, if there are another say k 7 is also if say k seven is also collating here if the k 7 then we have a k 7 over here like this. So, if a slot is containing more than two keys I mean more than one keys, then we have to use this chain we have to use the linked list for this. So, this is the collision. So, say for example. So, now, this chaining is a method to handle the collision. So, now, we want to analyze this chaining method how good this is ok.

(Refer Slide Time: 18:55)



So, this is the analysis of chaining. So, chaining is the method to handle the collision. So, now, suppose there are m slot 0 to m minus 1, and suppose there are this is the k the suppose there are k keys n keys there are n slot m n keys. So, so m is the number of slots and n is the number of keys. So, now, it is a k 1 k 2 k n. So, now, among this if there is a collision suppose this slot i th slot having collision say 52 say 55 96 like this.

So, all of this h of 52 is equal to i is equal to h of 55 these are the key value h of 96 all are mapping to same slot. So, this is basically the chain in that slot if there are say 3 keys are collating there. Now what is the worst case of this, now how to search key? Suppose we want to search a key suppose we want to search say 96 is there or not. So, what we do? We apply the hash function on it. So, it will it will first map to the i th slot then we know there is a chain.

So, we have to we have to read the chain basically. So, basically we are reading the we are just scanning the chain. So, that is the way we search a key. Now what is the worst case for this chain? So, worst case is now suppose all these elements are collating in a single slot. So, there are n keys all are collating here then this is the worst case then the search time will be order of n because if it is collating in to this slot then we have to because all are collating in the same slot. So, there is a chain of size n ok.

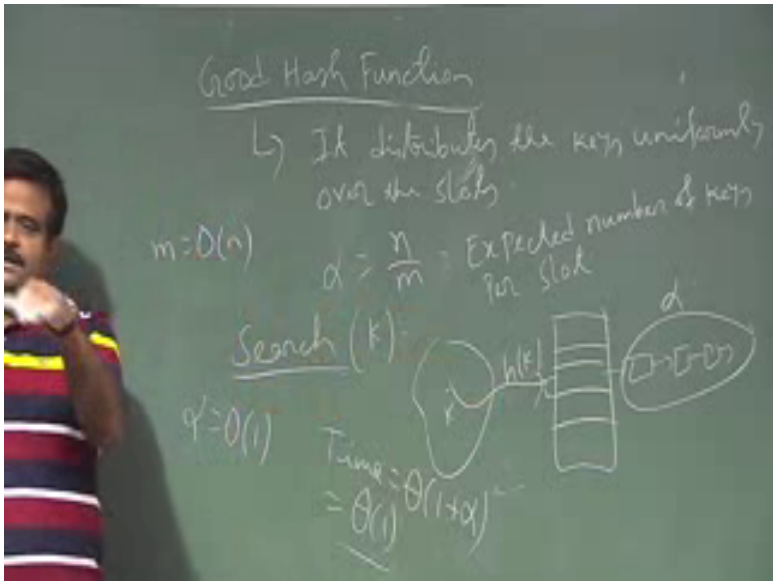
So; that means, when you search if it is if that key is collating here we need to search this whole list and this list is not sorted we are not going to sort this list then it will take some

more time to sort. So, it is just a unordered list linked list. So, we need to search our key in this linked list. So, that will take linear time that will take the time of the depending on the size of the list if the size of the list is linear the time complexity is linear. So, that is a bad hash function. Bad hash function in the sense that everybody is collating in the same slot.

So, there are n keys all are collating in the same slot that is a bad hash function. So, what is the good hash function? If we have uniform distribution of the keys over the slots, so, there are n slots. So, if n slot is distributed over this if there are n keys, if n keys are distributed over the slot uniformly. So, then n by m is basically what n by m is basically. So, there are n keys n slot if there are 100 say keys and if there are 10 slot. So, n by m is 10.

Basically then 10 is the number of keys per slot expected number of keys I mean or the. So, this is basically the number of keys per slot. So, this will happen if our hash function is such that it is distributing the key over the slot uniformly, it is not that all the keys are going to a single slot it is just we have n keys it is distributing the keys among the m slot uniformly. So that means, each slot will get n by m keys. So, this is called load factor this is referred as alpha this is called load factor ok.

(Refer Slide Time: 23:48)



So, this is a condition this is a criteria of a good hash function. So, a good hash function should such that it should distribute the key. So, it should. So, it distributes the key

distribute the keys uniformly over the slot. So, there are n keys m slot. So, each slot; that means, each slots should get same number of keys i mean the distribution is. So, given a key, it will be in one of this slot its probability is 1 by m it is equally likely.

So, then this alpha is the load factor n by m, alpha is the expected number of slot or expected number of keys per slot. So, in that case if we have such a hash function in that case search time will be how much. So, for search. So, you want to search a key. So, to search what we do. So, this is a say key we are going to search. So, we first apply the hash function on this. So, this will map to some slot i th slot ok.

Now, we know in the i th slot there is a linked list or chaining and this size of this chain is alpha. So, we have to just scan this. So, what is the time complexity for this? So, time is basically 1 plus alpha. Since one is the time to apply the hash function and then this alpha is the basically the load factor; that means, the number of keys per slot now if our hash function is a good hash function then this is the scenario and now if alpha is order of one.

So; that means, if the n is order of m or n is order of n then alpha is 1 then this will be constant time. So, if our hash function is a good hash function in the sense that it distribute the key uniformly over this slot then alpha will have a alpha is the expected number of keys per slot then this is the time for searching a key or insert a key because we first apply the hash function this will take one, and then plus alpha is the size of the list we have in that particular slot.

So, this is the idea of the hash function and this is collation is there and collation can be handle by the chaining, and next class will talk about how to construct such hash function. So, that it will be distribute the key uniformly over the channel. So, while we construct the hash function this should we should keep this in our mind that it should distribute the keys uniformly over the slots.

Thank you.