

Data Structures and Algorithms using JAVA
Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 48
Non-linear Searching Algorithms

As we have learned about linear searching algorithm when data is stored in an array and or in a linked list that is a linear data structure. Now there is another different searching algorithms also known when data is not stored in an array rather it is stored in some other data structure maybe tree. Now we will discuss one such technique here. This discussion related to that kind of search technique, it is called a nonlinear searching algorithm because data is stored on a nonlinear data structure like tree.

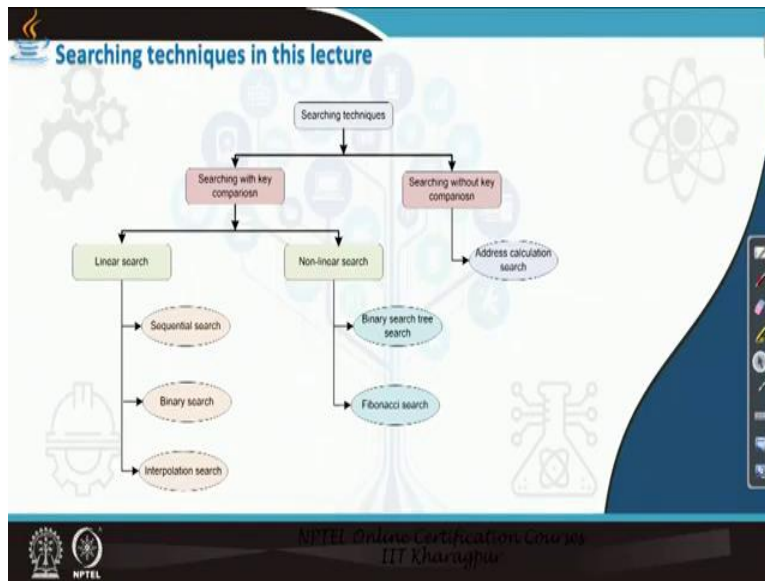
(Refer Slide Time: 1:04)



So the topic that we are going to discuss basic idea about non-linear searching algorithms, so when data stored in a tree actually we will consider when the data is stored in the form of a special binary tree called binary search tree how the searching can be performed? And then we will discuss about Fibonacci search. Fibonacci search is basically one idea about a special tree.

It is again binary tree but it is special tree called the Fibonacci tree. And then finally we will discuss about searching with hashing. Hashing although it is not non-linear data structure but it is also can be considered as a mixed up this one. So we will discuss about searching with hashing.

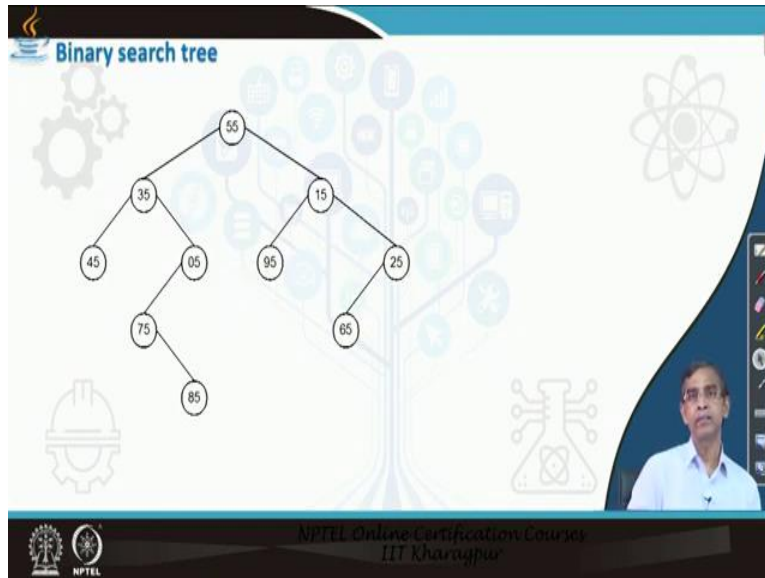
(Refer Slide Time: 1:51)



So this is the plan of the topic for today. So today actually we will try to cover these are the different techniques. These are the technique already we have discussed in the previous lectures. Now let us start about. First let us talk about the binary search tree based searching.

(Refer Slide Time: 2:14)





Here as the name implies that we store the data in the form of a binary tree rather we can say that binary search tree. Now we know exactly what is the property that a binary search tree suit, satisfy. The binary search tree is a special binary tree, I do not want to discuss it again. So all the elements are stored in where nodes, stored the value satisfying certain properties is called the binary search tree property. This is an example of a binary search tree.

Now we want to find an element. So finding an element is very easy while we are discussing binary search tree that time I gave an idea about that how searching can be carried out for this data structure. The same concept it is here also, only the thing is that from the given input list we have to store the list into the form of a tree.

That means the data is stored in the form of binary search tree rather and then you can just see exactly target key if you want to find whether the node, I mean that key or that value is present in any node in the binary search tree or not. That concept basically to be followed. So merely it is very searching. Now, if for example, we want to search for say 80. So what you can do is that 80 should be along this direction.

So you can come here. Anyway this is actually not a binary search tree ok. This is not a binary search tree. This is simply a binary tree. Now again let us see as it is not a binary search tree still if the element is stored in a binary tree whether you can apply the binary search technique or not, you cannot, in that case we can perform any tree traversal algorithm in order, pre-order or post

order any transversal algorithm that means traverse in particular order. And we have to continue this traversal until we find the element.

(Refer Slide Time: 4:08)

Searching with binary search tree: Algorithm

Input: A binary tree with *PTR* as the pointer to a node and *K* is the element under search.
Output: Returns 1 if *K* is available, otherwise returns NULL.

```
1 ptr ← PTR // Start from the current node of the binary tree
2 If (ptr = NULL) then
3   If (ptr → DATA = K) then // Visit. Check the element in the current node
4     Return (1) // Return the status of search
5   Else
6     BinaryTreeSearch (ptr → LC) // Then search left sub tree in preorder
7     BinaryTreeSearch (ptr → RC) // Then search right sub tree in preorder
8   EndIf
9 EndIf
10 Stop
```

Note:

- Algorithm follows preorder traversal and is defined recursively.

The slide also features a binary tree diagram with root 65, left child 35, and right child 18. Node 35 has children 45 and 95. Node 95 has children 75 and 85. Node 18 has children 95 and 25. A red arrow points from line 1 of the code to the root node 65. The slide includes NPTEL logos and a video feed of a presenter in the bottom right corner.

Now if it is in binary search tree that procedure can takes fewer amount of less amount of time. Here we have given the idea about how the searching with binary search tree can be carried out. Again this is not a binary search tree, by mistake it is coming. So this 15 should not be like this one anyway. So this tree is not correct actually, that is what I want to say.

But this is the algorithm that you can follow that the how the searching can takes place. Now if you follow this algorithm, then you can note that we follow basically a preorder traversals to search the tree actually. However, you can follow any other such algorithm, traversal algorithm like in order or post order also can be equally applicable.

So you have to just transverse its nodes and this traversal will continue until we finish the visiting all nodes or we find our target elements into the tree. So this is a basically recursive search algorithm. And few lines of code is required. You can write the program easily in Java programming language. It is very easy. So this is the idea about the binary search tree technique.

(Refer Slide Time: 5:13)

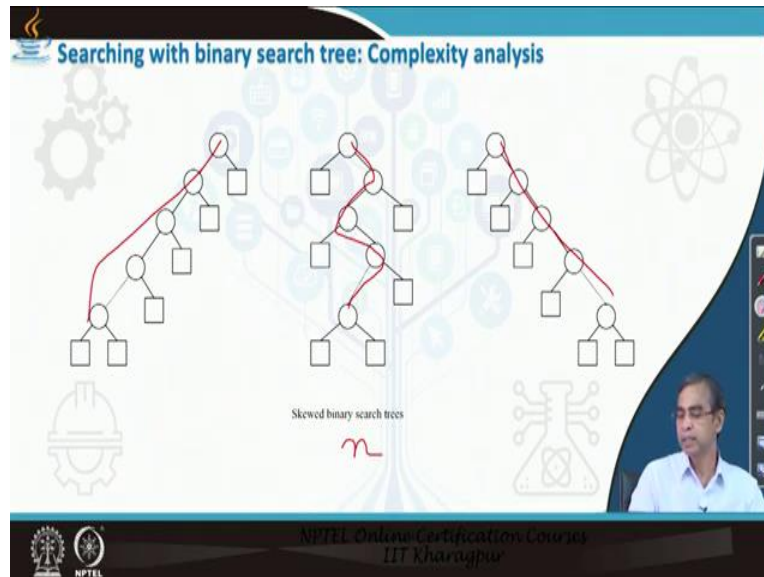
The slide is titled "Searching with binary search tree: Complexity analysis". It contains two diagrams of binary search trees. The left diagram shows a complete binary tree with 15 leaf nodes (represented by squares) and a height of 4. A red line traces a path from the root to a leaf. Below it is the text "Binary search tree with minimum height" and the handwritten expression $\log_2 n$. The right diagram shows a skewed binary tree with 15 leaf nodes and a height of 15. Below it is the text "Binary search tree with an arbitrary height" and the handwritten expression $\log_2 n < n$. The slide also includes a video feed of a presenter in the bottom right corner and NPTEL logos in the bottom left.

Now let us come to the complexity analysis. Now the complexity analysis depends on how your structure of the tree is. Now depending on the pattern of the data that is stored in a tree, a binary tree can be like this, then it is called a complete binary tree. And you know complete binary tree is a tree with n number of nodes, always gives the tree with minimum height.

If it is not complete binary tree then the heights will be greater than n . So sorry, if it is a complete binary tree then height of this binary tree will be in the order of $\log_2 n$ that basically discussed in previously. And if it is not complete binary tree then height will be ok less than n but it is $\log_2 n$ greater than $\log_2 n$. So this is the height which will be less than n or greater than n .

So this means that here the so now again why I am telling height because to search for an element maximum you have to come from this one. So this is the height of the tree actually, any path. So this is the $\log_2 n$ what is called movement is required or comparison is required that is why the complexity is $\log_2 n$. Now n that is the worst case situation. It is not obviously n , the worst case situation I can tell you when the binary search tree take the size skewed form.

(Refer Slide Time: 6:50)



So these are the some example of the tree where the height is maximum. So in this case, height for all tree is basically n where n number of stores, n number of nodes are there. Now so these are the maximum height that is possible. Now what you can say that if you follow binary search tree for your searching or if you can store the data in the form of a binary search tree and then you apply searching then your complexity will be, the lowest complexity that is possible is $\log_2 n$ that is the best case we can say and worst case it will take n , which is basically similar to the linear search technique actually.

And on the average it will be in between $\log_2 n$ and n actually. So this way this algorithm as the complexity in between the linear search when data stored in a linear array or it is in a binary tree. So it is a in mixing result actually. Now it depends on size of the structure of the array of course, how it basically stores the data. Now so this is the case that you have study about, this is a binary search tree technique.

(Refer Slide Time: 7:54)



Now let us discuss about another technique it is called Fibonacci search. As the concept of Fibonacci is not known to you, so I just want to give, take some time where I want to discuss about what is the Fibonacci concept is there. So a series of number can be called in a Fibonacci sequence if they satisfy certain property. The property is that the last element or any i th element is a sum of the previous two elements. If it follows this property then a sequence of numbers will be called a Fibonacci sequence.

(Refer Slide Time: 8:36)

The slide is titled "The concept: Fibonacci sequence". It defines the Fibonacci sequence and provides the recurrence relation: $F_n = F_{n-1} + F_{n-2}$ with $F_1 = 1$ and $F_0 = 0$. An example table shows the sequence from F_0 to F_8 . The values are 0, 1, 1, 2, 3, 5, 8, 13, and a blank space. The last two cells are highlighted with red boxes. To the right, there is a handwritten red box containing a question mark and the label F_n . A small video inset in the bottom right shows the same presenter. The footer includes the NPTEL logo and "NPTEL Online Certification Course IIT Kharagpur".

Fibonacci sequence:

- The n -th Fibonacci number in a binary Fibonacci series is given by

$$F_n = F_{n-1} + F_{n-2} \text{ with } F_1 = 1 \quad F_0 = 0$$

Example:

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
0	1	1	2	3	5	8	13	

Now here is an example as I told you, this is basically how you can say that nth Fibonacci number as you can say nth Fibonacci number means any Fibonacci number is basically sum of the n minus one Fibonacci number and n minus two Fibonacci number. And this is a boundary condition first Fibonacci number, and then this is the first Fibonacci number and this is the second Fibonacci number, it is zero and one.

So here, for example, these are zero and one. The next is basically sum of the two. So this is one. Then next is basically sum of the two so it is this one. Then this one is the sum of these two. So three one, this one is basically sum of these two, five and this element is sum of these two. Now you can see the next Fibonacci number say F7 you can easily calculate that this will be 13. And likewise, if you continue so eight Fibonacci number can be calculated this way.

And if we continue this, any nth Fibonacci number can be calculated this one. Now, so any Fibonacci number we will be able to calculate. So this is a series, this is basically a series with n what is called the numbers in the series and they follow certain properties and such a series is called Fibonacci series or called Fibonacci sequence. Now this is the concept that is basically followed and this concept can be used to build a tree called the Fibonacci tree.

(Refer Slide Time: 10:14)

The slide is titled "The concept: Fibonacci sequence". It defines the Fibonacci sequence as the n -th Fibonacci number in a binary Fibonacci series is given by $F_n = F_{n-1} + F_{n-2}$ with $F_1 = 1$ and $F_0 = 0$. An example table shows the sequence: $F_0=0, F_1=1, F_2=1, F_3=2, F_4=3, F_5=5, F_6=8$. A diagram illustrates a Fibonacci tree where F_6 is the root, branching into F_5 and F_4 , which further branch into F_4 and F_3 , and F_3 and F_2 respectively. A small video inset shows a man speaking.

Now idea it is like this say suppose any ith Fibonacci number, so ith Fibonacci number is basically can be calculated if we can calculate the Fi minus 1 because in order to calculate the ith

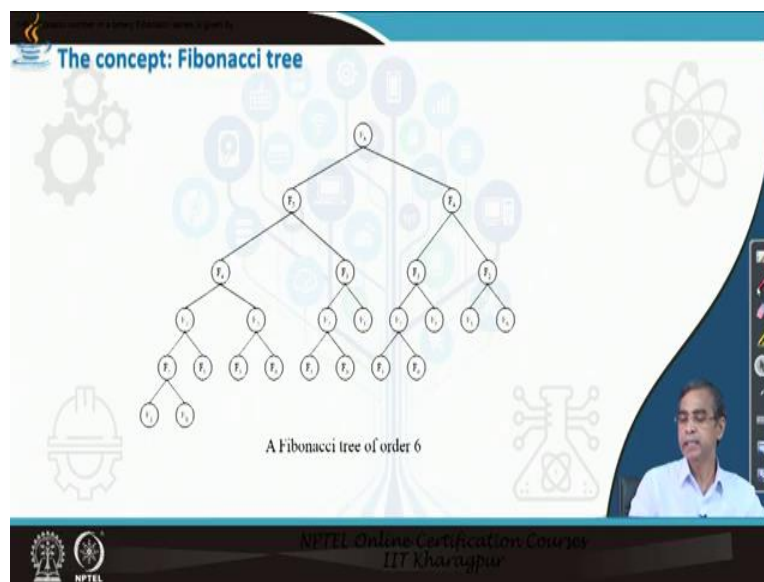
Fibonacci number, we know F_i minus one and F_i minus 2 or we can say this is two and this is F_i minus one. Now actually you know, if I ask you to calculate i th Fibonacci number, for example, the last slide we told about n th Fibonacci number.

Because in order to calculate the n th Fibonacci number you know what is the n minus 1 Fibonacci number and what is the n minus 2 Fibonacci number. Now this basically calculation can be easily done by writing a recursive program. So a recursive function can be planned so the n th Fibonacci number.

Now if you draw the recursion is basically have a recursively like. Now this F_i minus 2, that means she is basically to calculate this one F_i minus three and this basically F , so you can say F_i minus 4 and F_i minus 3. Now again we can split this one. Then again, we can F_i minus 5 and F_i minus 6 and so on. So it will go on.

So these basically called the recursion, the way the recursive calculation will takes place. Now you see these recursive calculation the way it takes place, it form a tree and it is the recursion tree and recursion tree for Fibonacci calculation. Now such a particular tree can be termed as a Fibonacci tree. Now here in the next slide, I can show you how a Fibonacci tree look like this.

(Refer Slide Time: 12:07)



So this is an example of a Fibonacci tree. And for the, this is a Fibonacci tree for the order six because the last, the root node is basically six Fibonacci that means if we want to calculate n th

Fibonacci number, we can say Fibonacci tree of order six. Now, so this is the idea about how the Fibonacci tree will look like. Now this is basically the recursion tree again, alternatively to calculate the F_6 that means sixth Fibonacci number like.

Now this information is very vital to perform these kind of tree into a searching which can give a better searching algorithms and this is also comparable to the binary search tree because binary search tree is good, but not always. In the worst case it take time, n . But here Fibonacci search tree whether the best case, worst case and average case it will take less time than the binary search tree actually that is why this is more preferable.

(Refer Slide Time: 13:08)

The concept: Fibonacci tree

F_0	F_1	F_2	F_3	F_4	F_5	F_6
0	1	1	2	3	5	8

A Fibonacci sequence of order 6

A Fibonacci tree of order 6

NPTEL Online Certification Courses
IIT Kharagpur

Now let us see how we can apply these binary search I, how we can apply the searching with the Fibonacci data structure. Now the idea it is that, I repeat it again for the Fibonacci tree of order six, the same thing but we have to do a little bit different what is called a modification into this tree. I will come to the modification. First of all the number that you have to store it is not that they are in a Fibonacci sequence, they can be anything.

So one is four, another is six, another is any number, so it can be eight, any order actually you can take not necessary to be in a sorted order, so five, one two and three so this is basically. So what we can this is basically the input list. All these input lists can be mapped to some Fibonacci

number. Then they can be stored in a Fibonacci tree actually and then we will be able to search depending on this one.

If we arrange this in an order it is better of course that will include performance of course, but even if it is not in order also no issue. So we can do that. Now let us see how we can map an elements into an i th Fibonacci or we can store this element into the Fibonacci tree and later on we can perform the searching.

That idea is can be explained like this little bit modification of this Fibonacci tree of order. This is the raw Fibonacci tree or basic Fibonacci tree we can say. We have to modify these Fibonacci tree in order to amenable to our such algorithm called the Fibonacci search. Now what modifications that is required I just mentioned two rules in this figure.

(Refer Slide Time: 14:53)

The concept: Fibonacci search tree

Rule 1:

- For all leaf nodes corresponding to F_1 and F_0 , we denote them as external nodes and redraw them as squares, and value of each external node is set to zero.

0	1	1	2	3	5	8	13	21	34
0	1	1	2	3	5	8	13	21	34

So two rules are like this. The first rule is that for all leaf node, you know these are the leaf node actually all the leaf nodes that is corresponding to F_1 and F_0 we denote them as external nodes. Probably the corresponding external nodes are familiar to you. So they are basically can be marked as a square nodes. So as you can see here.

So these are the external nodes are there and value of the external nodes can be made zero. So here we see, these are the all external nodes and we made the value is zero. So this is the one

modification that we can do it. This is the first rule and then second rule, second rule is basically another modification of the internal nodes.

(Refer Slide Time: 15:41)

The concept: Fibonacci search tree

Rule 2:

- For all nodes corresponding to F_i ($i > 2$), each of them as a Fibonacci search tree of order i such that
 - the root is F_i ,
 - the left sub tree is a Fibonacci search tree of order $i-1$,
 - the right sub tree is a Fibonacci search tree of order $i-2$ with all numbers increased by F_i .

Fibonacci tree with Rule 1

Fibonacci tree with Rule 2

NPTEL Online Certification Course
IIT Kharagpur

So the second rule can be stated like this. The second rule can be stated like this, so this is a rule two rule. Rule two says that the root it is an F ith number, the left sub tree is a Fibonacci search tree of order i minus one so this is F_5 . Now here the right sub tree we have to little bit modify. The right sub tree is a Fibonacci search tree of order i minus two with numbers. This is important increased by F_5 .

So for example this is a left. So what we can do is that the left sub tree will be increased by, this is basically this is a left sub tree will be increased by its parents so it is basically 7. And if it is 7 then it is 7 and this will be 7. So this basically we are in, so all the nodes of the left sub tree will be increased by its parent value. So this way if we continue, then you can say that, so this is basically 8, this is at left so you can increase by the value.

Earlier it is 2 then it is five so parent is increased so this one and this one, one so parent will increase 6 and this. So again if you consider this one it will be seven and it will 6 earlier it is there. Now and then again come to here. It is basically same as this is earlier and then we will increase by this one. So this procedure if we apply to all then we will be able to modify the algorithm this one.

those are basically 15, everything can be stored as an internal nodes. So this way you can store the internal nodes. Then we can take the searching.

Now depending on the element that here so we have to go to the left or right, depending on the target element. So it will go to the, this one or these sides depending on target element. But here it is not exactly binary splitting as it takes place in case of binary search tree. But it will take place in a Fibonacci level like, so Fibonacci order actually. So this way the searching can be taken place.

(Refer Slide Time: 19:28)

```
/* Initialization */
1 i ← Fn // Start at Fn-th location
2 p ← Fn-1, q ← Fn-2 // Pointers to the left and right sub trees of the node Fn
3 If (K = Ki) then // Compare the key K with the value at i-th location
4   If (q = 0) then
5     Print "Unsuccessful"
6     Return() // Search is over
7   Else
8     i ← q, p ← p - q, q ← p - q // Go to left sub tree
9     Go to Step 3
10  EndIf
11 EndIf
12 If (K < Ki) then
13   If (p = 1) then
14     Print "Unsuccessful"
15     Return() // Search is unsuccessfully terminated
16   Else
17     i ← q, p ← p - q, q ← q - p // Go to right sub tree
18     Go to Step 3
19   EndIf
20 EndIf
21 If (K > Ki) then
22   Print "Successful at i-th location"
23 EndIf
24 Stop
```

Here is an example that is given here. And this is an algorithm that you can follow to write your code. That means how you can build up Fibonacci search tree and then finally you can perform the Fibonacci search.

(Refer Slide Time: 19:38)

Searching algorithm with Fibonacci search tree: An example

Input list:

0	1	2	3	4	5	6	7	8	9	10	11					
15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95

Suppose we want to search for $K = 25$

Initialization

$i = F_3 = 3$ $p = F_{i-1} = 5$ $q = F_{i-2} = 3$

Iteration 1

$K_i = A[i] = 50$
 $K < K_i, q = 3 \neq 0$
 $i = i - q = 3 - 3 = 0, p_{old} = p = 5$
 $p = q = 3, q = p_{old} - q = 5 - 3 = 2$

Iteration 2

$K_i = A[i] = 35$
 $K < K_i, q = 2 \neq 0$
 $i = i - q = 3 - 2 = 1, p_{old} = p = 3$
 $p = q = 2, q = p_{old} - q = 3 - 2 = 1$

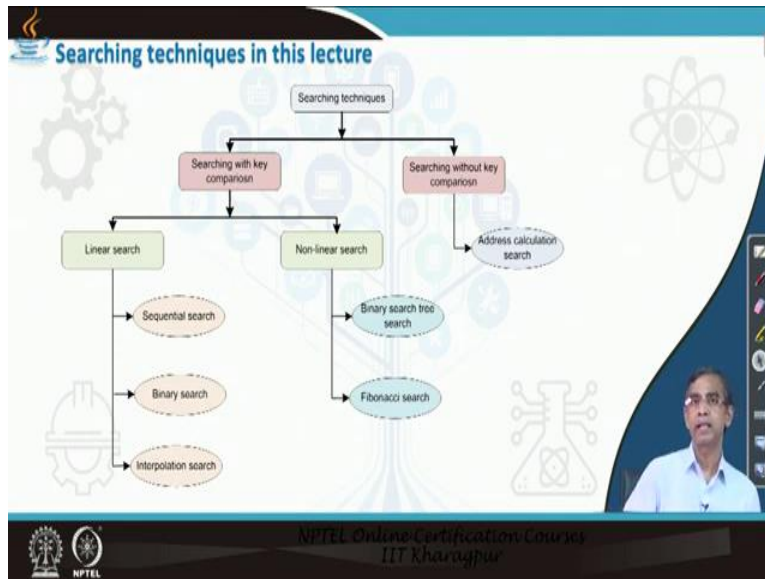
Iteration 3

$K_i = A[i] = 25$
 $K = K_i$
Search is successful

NPTEL Online Certification Course
IIT Kharagpur

And this is a simple example that you can give and suppose k is 25 and what are the different iteration that is required? You can follow this one. So this way you will be able to search it very efficiently. And the complexity of this searching will be $\log_2 n$ allowing whether worst case or best case because it is a, that fashion only because the binary, the tree is a binary tree of course that so have the height not exactly the complete binary tree but it is in between complete binary tree and then worst case are always better than the worst binary tree actually. So this is about the Fibonacci search.

(Refer Slide Time: 20:17)



For the details about this searching technique again, I can suggest you to go to the book where you can find a detailed discussion and then coding and everything there.

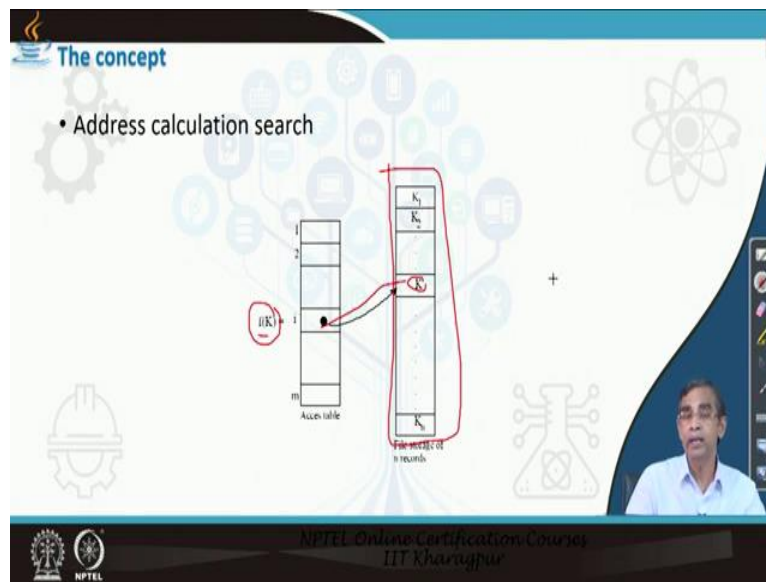
(Refer Slide Time: 20:28)



Now I will discuss about the next technique, it is called the hashing technique. Hashing technique is basically one technique without key comparison. Here we do not have to compare key. If you see all the searching algorithm here it basically, we have to compare your target key with the key that needs to be matched or not matched, whatever it is a search. So comparison is

required but here no comparison is required. Then without any comparison how the hashing can be, how it can be possible? So we can apply the hashing technique.

(Refer Slide Time: 21:01)



So we have little idea about the hashing technique. So here basically idea is that if suppose these are the, these are the suppose your elements stored, these are the elements that means we want to search the item from here. They are basically keys we can say. Then there is basically one function that we have to calculate. It is called the hashing function.

Hashing function can tell you in which location this K is stored in an array like. So this basically stores that index location it is stored. So this F key if we apply on a element then it will basically give you in which location. And that is how this technique is called the address calculation search because we have to calculate where the element is located.

(Refer Slide Time: 21:52)

The concept

- Address calculation search

Accessible

Table address of n records

NPTEL Online Certification Course
IIT Kharagpur

Now let us proceed about this technique. Actually here the problem that matters or the main task it is there how we can decide this address calculation, how this f_k function can be that means f is a one formula or a function we can say which is basically take an input as a key and it will return the address of the location of the key. So how this formula can be formulated or this function can be formulated. So this is the only task it is there.

(Refer Slide Time: 22:29)

Hashing mechanism

Hashing: Is a mapping from key to its index (location)

$H: K \rightarrow i$

K	i
10	1
25	2
35	4
43	7
62	8
59	4
31	4
49	3
77	1
33	6

0	19
1	10
2	
3	49
4	69, 31, 77
5	
6	33
7	43
8	35, 62
9	

NPTEL Online Certification Course
IIT Kharagpur

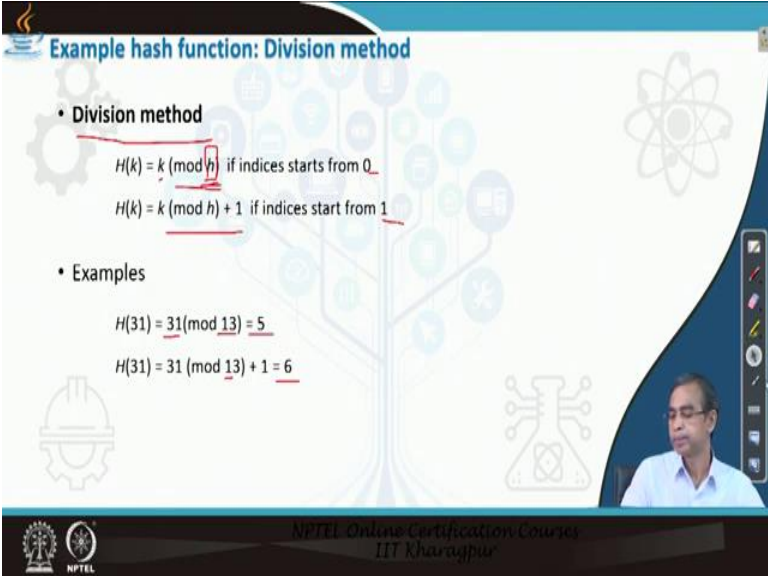
Now regarding these things, there are many, many hashing technique is proposed is basically map that is why it is called mapping also and that from a key to a location. So that mapping is basically just like a mapping for every key it is there. Now here is an example. These are the basically key is the location, key the element to be searched.

And these are the basically different, some functions you have used which basically says that location of these element 10 is 1, location of this 19 is 0, location of this element is say 8. So some formula you have used and this basically gives the formula it is there. So possibly you can say that which technique it is there anyway.

So what actually, if you do it, then we can say all the key those are basically to be stored in this array within this suppose size 10, then some element has to be stored in the same location because of the same address is written by this one. But this is not a good idea of hashing actually.

It should give the different location possibly then it will give the better idea because here for 59 31 and 77 they are the three different keys but it basically return you the same location, same hash location we can say. So that is why this hashing technique that I have illustrated is not necessarily the good one.

(Refer Slide Time: 24:06)



Example hash function: Division method

- **Division method**
 $H(k) = k \pmod{h}$ if indices starts from 0
 $H(k) = k \pmod{h} + 1$ if indices start from 1
- **Examples**
 $H(31) = 31 \pmod{13} = 5$
 $H(31) = 31 \pmod{13} + 1 = 6$

NPTEL Online Certification Courses
IIT Kharagpur

Now let us see how the different hashing techniques that is known in this field actually. There are many hashing techniques are known. Some techniques are called based on division method.

Every technique has their own advantage as limitation, pros and cons is there. So division method says that if the key is the target element, then H is one value to be decided a-priori that means it is your decided. Usually people think that this h should be a prime number, maybe 11, 13, 17, 19 these kind of things are best values to give it.

And here so if your address search from zero then you can follow this one. If it is search from one then you can follow this one. Here is an example. We take the 13 as the hash value, so h is called hash value and hash function that if we apply on a particular element say 31 it return 5 on 13. If it is 30, if it is 31, if it is the address location start from 0. If it is, address location start from one then it is six. So this way the elements can, the this can be calculated.

(Refer Slide Time: 25:31)

Hashing mechanism

Hashing: Is a mapping from key to its index (location)

Diagram illustrating the mapping from key (K) to index (I) via a hash function $H: K \rightarrow I$.

Key (K)	Index (I)
10	1
19	0
35	8
43	7
62	8
59	4
31	4
49	3
77	4
13	6

Index (I)	Value
0	19
1	10
2	
3	49
4	59, 31, 77
5	
6	33
7	43
8	35, 62
9	

NPTEL Online Certification Course
IIT Kharagpur

Now again in the previous slide that we have given their, previous slide let us come to the previous slide. Here if we apply the same hash function, same hash function on this element, then you will be able to calculate address, then you can use it and then you can check it how it works. So this is the idea about that you can think about.

(Refer Slide Time: 25:48)

Example hash function: Mid-square method

- **Mid-square method**
k : 1234 2345 3456
k²: 1522756 5499025 11943936
H(k): 525 492 933
- **Limitations**
 - Computationally expensive

NPTEL Online Certification Course
IIT Kharagpur

Now let us consider a few more methods in this direction. So earlier the method is called division method and the next method is called the mid square method. A mid square method, I just give some example, say suppose k is 1, 2, 3, 4. It basically, the idea is that you pass on the square of this one and then take the middle values for example, take the middle fifth to say, no alternative value. So this will give you five two five with the address value.

Now again repeat the same thing 2, 3, 4 and square it and then taking the mid values. Then it will take this one and it continuing this way. So this is the idea so this is the key values. And this is a technique that I told you, this technique is not a simple function but it is a code actually because you have to square and then take the alternate value, digits from the numbers, then it basically gives you the address.

Now, this is better compared to the previous method because it has the collision probabilities less. But collision is always there in the previous case also there, here, also there. But this method is computationally expensive because multiplication operation is involved. Then you have to find the alternate digit. So it is comparatively expensive compared to the previous method. But it is one method which is also as a possible method that you can apply to calculate the hash address of a number.

(Refer Slide Time: 27:18)

Example hash function: Folding methods

- Folding Method**
 $H(k) = k_1 + k_2 + \dots + k_n$
- Example**

Method	Input k	Chopping	Pure folding	Fold shifting	Fold boundary
1	0132756	01 52 27 56	$01 + 52 + 27 + 56 = 136$	$10 + 52 + 72 + 56 = 190$	$10 + 52 + 27 + 65 = 154$
2	0549025	05 49 90 25	$05 + 49 + 90 + 25 = 169$	$50 + 49 + 09 + 25 = 133$	$50 + 49 + 90 + 52 = 241$
3	1194936	11 94 39 36	$11 + 94 + 39 + 36 = 180$	$11 + 94 + 93 + 36 = 234$	$11 + 94 + 39 + 63 = 207$

NPTEL Online Certification Course
IIT Kharagpur

Now another is called the folding method. So folding method says that the key that you have to take into account you first divide into few parts. For example, if this is the input key, so divide into this part like or you can say first in a two digit like and then this one right and then and this is 0. So you can see this one.

Now for example here this one, this one and this one so here also you can say this on, this one, this one. So it is called folding that means the elements can be just partitioned into two digit each. Then what you can do is that we can take the sum of all the partitions number that is there for example here 136, it is 169, it is 180. So such a folding method is called pure folding.

Another method is called fold shifting. What is the idea is that in this case you do the same thing here like this one but just, reverse the previous one. So it is basically 10 other you can and then alternative partition can be previous. So 27 is 72 and this one. Here for example, this become this one and this becomes this one and this remain same and then we can find it, so this is the hash values and like one.

Fold boundary it is basically alternatively. Fold boundary only the, this one and this one. So here this one and this one then you take it. So, whatever you take you can see, you can find for a given key some address you will be able to calculate. But the problem here is that it is obviously

comparatively less expensive than the previous method, the square, the mixed square folding method.

But here the idea is that here the numbers that needs to be considered should be a little bit larger one because here you can see as very larger, if the number consists of long integers like or digits then it is good, but it is also suffer from the coalition. Coalition, actually there are no any hashing technique that you can think for, which is absolutely coalition free.

(Refer Slide Time: 29:40)

Collision resolution techniques

- Closed hashing
 - also called *linear probing*
- Open hashing
 - also called *chaining*.

NPTEL Online Certification Courses
IIT Kharijpur

Now, so for the coalition is concerned there are two ways this coalition can be addressed. One is called the closed hashing and another is called open hashing. Closed hashing means if you find a coalition, then suppose you have to store hundred elements into an array, so hundred location is given to you but because of the coalition two or more keys can come to the same location. But hundred elements to be stored hundred locations is there.

So if you store more than two numbers into same location, so some locations will be made vacant actually. So what actually idea is that if you find a coalition then you have to go little bit less to search that if there is a vacant position then place it there. So this way if a hundred elements to be stored and if you generate even the address space which is less than hundred but still it can occupy. And this kind of technique is called the closed hashing. On the other hand the linear hashing, linear hashing technique this is the linear probing actually it is called.

(Refer Slide Time: 30:49)

Closed hashing

- The search will continue until any one of the following cases occurs:
 - The key value is found.
 - Unoccupied (or empty) location is encountered.
 - It reaches to the location where the search was started.

NPTEL Online Certification Courses
IIT Kharagpur

And another probing is called the open chaining.

(Refer Slide Time: 30:52)

Closed hashing

- Example
 - Input: 15 11 25 16 9 8 12 8
 - Hash function: $H(k) = k \bmod 7 + 1$

Diagram illustrating the insertion process into a hash table of size 10 (indices 1-10):

- Initially the hash table is empty.
- Insertion of 15: 15 is hashed to index 2. Index 2 is empty, so 15 is inserted at index 2.
- Insertion of 11: 11 is hashed to index 5. Index 5 is empty, so 11 is inserted at index 5.
- Insertion of 25: 25 is hashed to index 6. Index 6 is empty, so 25 is inserted at index 6.


NPTEL Online Certification Courses
IIT Kharagpur

This is the example of closed hashing that I have told you. You just these are the elements you just consider. And this is the hash function. You see the coalition is coming. Whenever coalition is coming, you have to go to the next free vacant position keep it there. So this way it will produce it will store all the elements in this array. So this is basically the idea of the closed hashing.

(Refer Slide Time: 31:16)


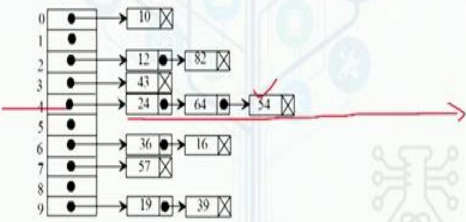
Drawback of closed hashing

- **Clustering**
 - key values are clustered in large groups and as a result sequential search becomes slower and slower.
- Following are some solutions known to avoid this situation:
 - Random probing
 - Double hashing or rehashing
 - Quadratic probing



Open hashing

- Also called separate chaining
 - This method uses a hash table as an array of pointers
 - Each pointer points a linked list
 - hash table is an array of list headers.



Now let us come to the problem that they, there is a many way this closed hashing can be resolved. One idea is called chaining. What is the idea? That you calculate the hash address but coalition may come. For example, here coalition is coming but whenever the coalition is coming so what we can do is that we can maintain a linked list.

And all the data, I mean same address can be stored into this linked list in the order they are basically calculated. Now the problem is that we do not have to go for this one and whatever it is

there and not necessary the address space should be same as the number of elements, it can be even less also. So this is one better idea than the previous one.

But one problem is that if you are not so much fortunate enough then all the elements can come to only one location and this list will go in this direction and it will become the single linked list storing all the elements and you know complexity will be very high. And here also absolutely you have to, whenever you have to find a target element calculate it and then come there. So you have to compare the key elements.

So it is absolutely not that without any key comparisons. So somehow it key comparison is coming into this picture. So it is an hybrid techniques, but it is it works better assuming that all the elements that you have to work with, they are basically uniformly distributed or scatterly distributed then this kind of technique is far better and then that is more preferable. So this is the different idea that you can think about hashing.

(Refer Slide Time: 32:52)

Advantages of open hashing

1. Overflow situation never arises. Hash table maintains lists which can contain any number of key values.
2. Collision resolution can be achieved very efficiently if the lists maintain an ordering of keys, so that keys can be searched quickly.
3. Insertion and deletion become quick and easy task in open hashing. Deletion proceeds in exactly the same way as deletion of a node in single linked list.
4. Finally, open hashing is best suitable in applications where number of key values varies drastically as it uses dynamic storage management policy.

NPTEL Online Certification Course
IIT Kharagpur

And then with this hashing we can store the data and then we can perform search and searching is far better than whatever searching technique that we have discussed about either linear search or non-linear search better. So hashing in that sense is better.

(Refer Slide Time: 33:04)

Analysis of hashing

$$\lambda = \frac{\text{Total number of key values}}{\text{Size of the hash table}}$$

$S(\lambda)$ = Average number of probes for a successful search.

$U(\lambda)$ = Average number of probes for an unsuccessful search.

$$U(\lambda) = (1-\lambda) \frac{1}{(1-\lambda)^2} + \frac{1}{(1-\lambda)}$$
$$S(\lambda) = \frac{1}{\lambda} \ln \frac{1}{1-\lambda} +$$

NPTEL Online Certification Course
IIT Kharagpur

The complexity calculation it is like this. If lambda is a factor which is defined this one total number of key values divided by total size of the hash table. This lambda can be greater than one, even less than one, whatever will the value it is usually should be less than one. Then we will be able to calculate about S lambda is average number of probes for a successful search.

U lambda is average number of probes for unsuccessful that this formula is given. So this basically tells the percentage search that is required in order to find this one. And so this calculation says that this searching algorithm is better compared to the other technique that we have learned in this category.

(Refer Slide Time: 33:45)

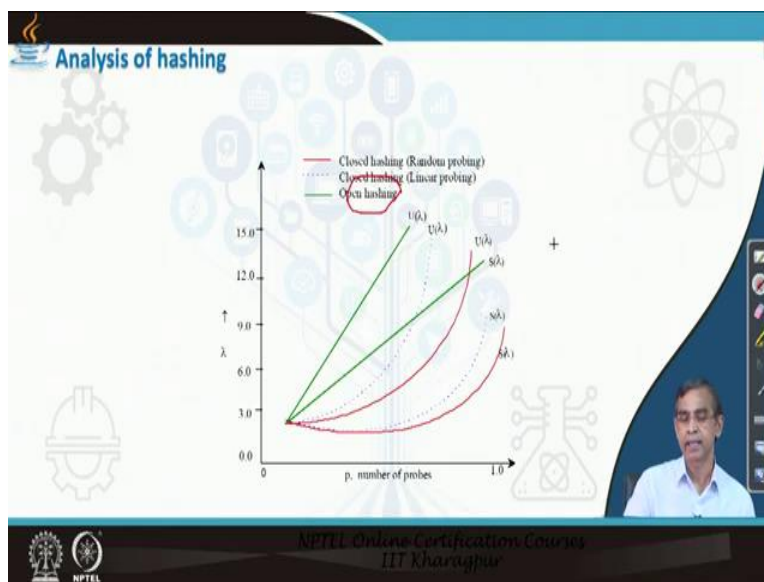
Analysis of hashing

- Closed hashing with random probing
$$U(\lambda) = (1 - \lambda) \frac{1}{(1 - \lambda)^2} = \frac{1}{(1 - \lambda)}$$
$$S(\lambda) = \frac{1}{\lambda} \ln \frac{1}{1 - \lambda}$$
- Close hashing with open probing
$$U(\lambda) = \frac{1}{2} \left(1 + \frac{1}{(1 - \lambda)^2} \right)$$
$$S(\lambda) = \frac{1}{2} \left(1 + \frac{1}{1 - \lambda} \right)$$
- Open hashing
$$U(\lambda) = \lambda \quad S(\lambda) = \frac{\lambda + 1}{2}$$

NPTEL Online Certification Course
IIT Khharajpur

So this is this is another for another time I have discussed about closed hashing and then chaining open hashing rather and the different analysis it is mentioned here. So this basically for the little bit analytical purpose so that how it works. A graph can be drawn giving plotting all these values then you will be able to understand how they can work it is there.

(Refer Slide Time: 34:12)

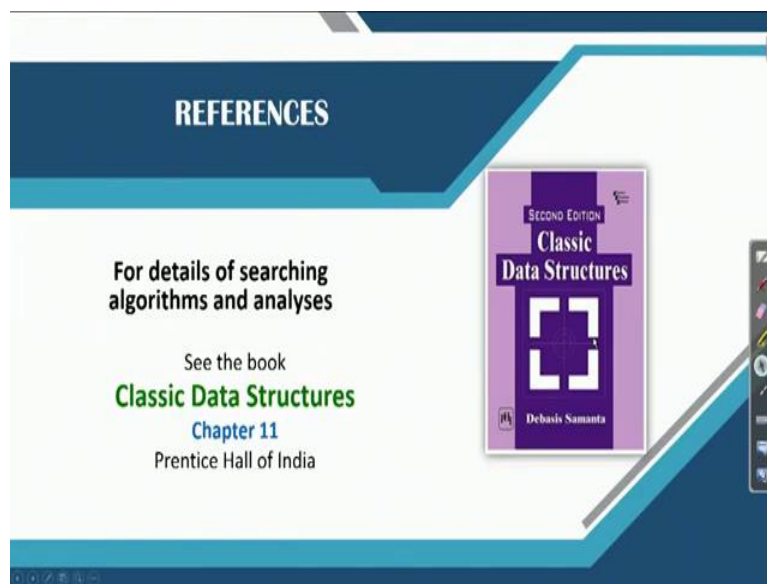


Now here is a graph you can follow. This graph actually shows how the different technique that works for you, now this is the green line that I have shown here. They are basically called open

hashing. Open hashing means using linked list that I told. And this is for unsuccessful, this is for successful. And other techniques are basically closed hashing using random probing and linear probing. Now if we consider the random probing then it will take this kind of complexity.

But as a number of probes increases the total rate of successful or unsuccessful also increases exponentially whereas for open hashing it is basically increases linearly. So this basically graph suggests that open hashing is a better one choice than the other closed hashing technique actually. Anyway so these are the different way the address calculation search can be done. We have given some idea about.

(Refer Slide Time: 35:14)



For details idea that you can follow from this book where the thorough discussion is available with a lot of illustrations and example so that you can follow this concept better consulting this book. Thank you very much.