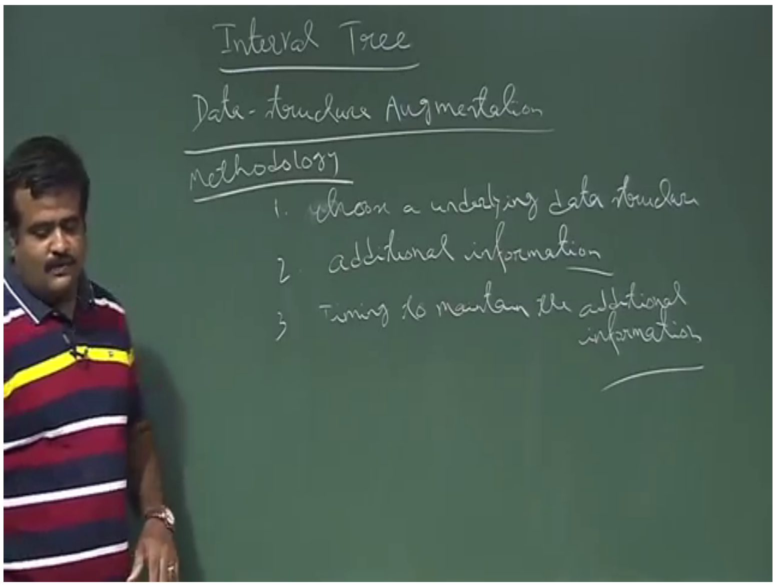
# An Introduction to Algorithms

**Prof. Sourav Mukhopadhyay**
**Department of Mathematics**
**Indian Institute of Technology, Kharagpur**

## Lecture - 30
## Interval Trees

So we are talking about data structure augmentation. So, we have seen the dynamic order statistics problem how we can use the data structure augmentation. So the so now, in this in this lecture we will talk about interval tree the. So, before the problem let us just recop the recap the data structure augmentation methodology.

(Refer Slide Time: 00:42)



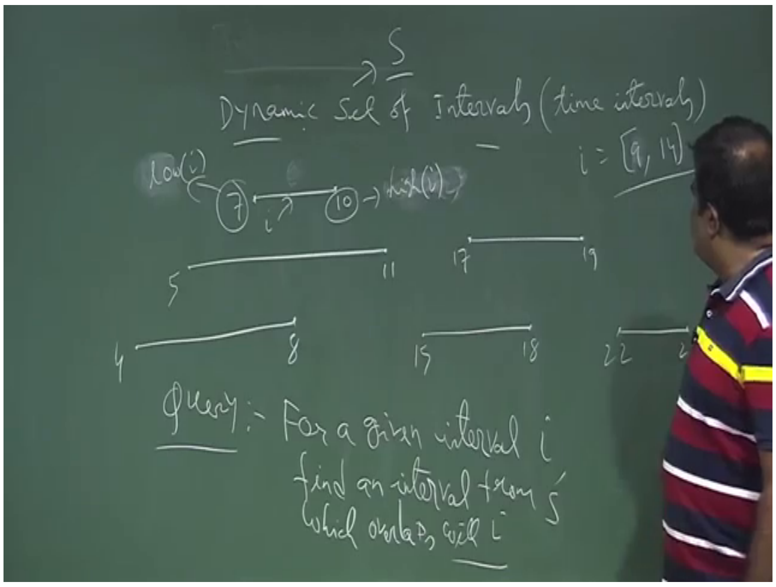
Interval Tree
Data-structure Augmentation
Methodology
1. choose a underlying data structure
2. additional information
3. Timing to maintain the additional information

So, the idea is so, idea is the methodology is basically so, we have to choose a underline data structure. So, we have to choose a underline data structure and then after that we have to determine the additional field of information we are going to keep for these data structure for our problem. So, additional information we want to keep. And then we have to see how this additional information can be maintained, when we do the modifying operation like insertion deletion if we choose the underline data structure as the red black tree like we did for dynamic order statistics.

And then the additional information we kept as the size of the subtree rooted at that node. And the we have seen that modifying operation insertion will not take much is similar type, similar time, same time as we do it for the red black tree. So, the additional information can be maintained with the same timing as the original data structure. So, the timing for timing to maintain the additional information. So, this is also crucial because if we spending much more time to maintain this additional information field then there is no use of these. So anyway so, this is the methodology behind the data structure augmentation.

Now, the interval tree problem is we have given some intervals there are basically time intervals and that set is dynamic set. So, any interval can join at any point of time and any interval can leave from any point of time.

(Refer Slide Time: 03:21)


S
Dynamic Set of Intervals (time intervals)
i = [9, 14]
low(i)
7
i
10
high(i)
5
11
17
19
4
8
15
18
22
Query :- For a given interval i find an interval from S which overlaps with i

So, so we have a dynamic set of intervals, basically time intervals and this set is dynamic. We have a dynamic set of intervals. And then now query suppose we have this interval say 7 10, 5 11 and say 17 19 and we have 4 8. So, these are the say 15 18 say 22 23. So, suppose we have a dynamic set of interval. So, this is a collection of intervals, each is the interval their time interval this is a I, I is a interval and this is this is left of I left endpoint. And this is 10 is the right of I right endpoint ok.

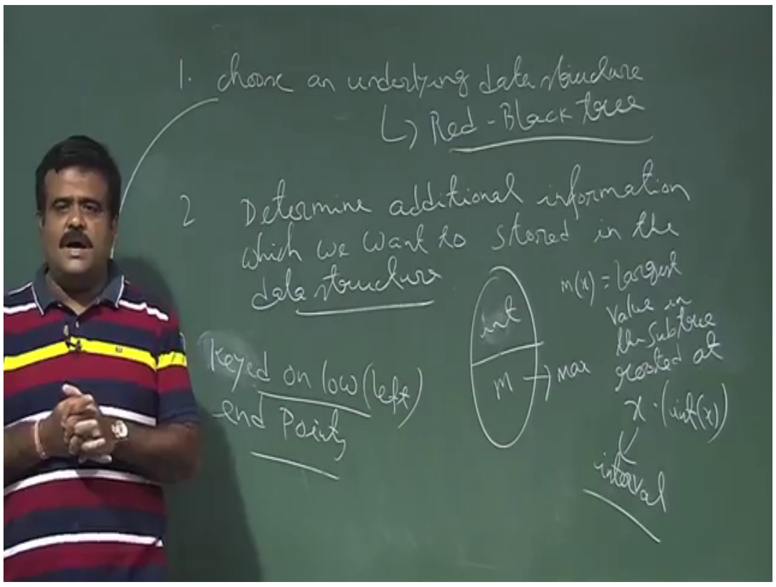
So, this is basically sorry low or high. So, low endpoint or this is the high end point. So, this is low an low endpoint and this is high end point. So, any interval has 2 endpoints low and high. So, this set is dynamic set. So, any point of time any interval can join and any point of time any interval can leave. So, this set is S set. So, this S set is the this set is basically this set is S, the collection of interval. And this set is dynamic set now we need to do this query like this, then for a given interval, for a given interval, say we have given interval I, we need to find an interval from S which is overlapping with I find an interval from S which overlaps with I. So, that is the problem. So, our problem is to the query is we have given a interval I. So, I has a low and high endpoint and then we need to find the interval which is overlapping with this intervals. So, that is the problem.

So, for example, if I is say we have say interval say 8 or 9 14. If I is 9 14 then we need to find an interval which is overlapping with 9 and 14 so.

9 and 14 is this 7 and 10 is overlapping with 9 and 14. So, we can just return 7 and 10. So, we thought this problem we are just looking for an interval which is overlapping with this. So, this is the problem. So, for this problem we need to maintain a data structure, we need to have a data structure to maintain this set S. So, what we do we need to do the data structure augmentation. So, so for that we will use a instead of having a new data structure or data structure from the scraps, we just use a underline data structure and we will do some augmentation there to act some new field. We store some (Refer Time: 07:52) here new information.

So, basically this is the methodology we choose and underline data structure.

(Refer Slide Time : 08:00)


1. choose an underlying data structure
↳ Red - Black tree
2 Determine additional information which we want to stored in the data structure
int
m
max
m(x) = largest value in the subtree rooted at x · (int(x))
interval
Keyed on low (left) end Point

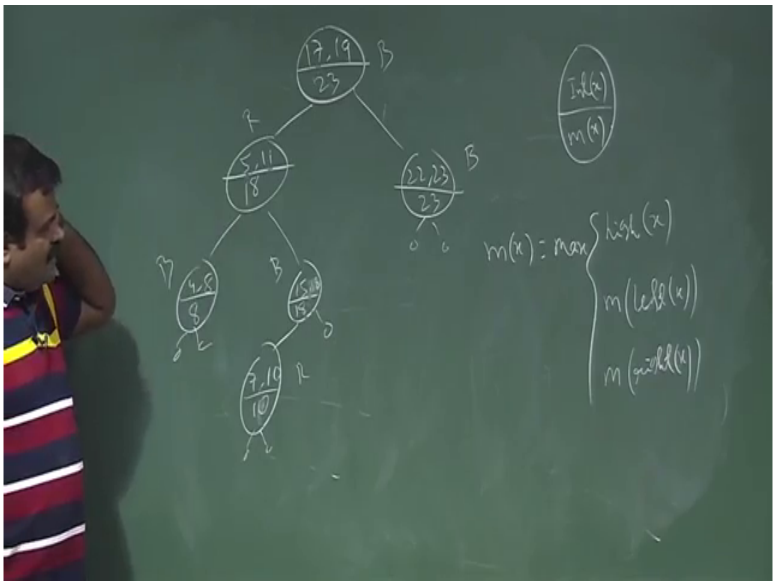
And here we are going to choose the red black tree, because red black tree is a balance tree. So, per performation anything can be done in logarithm time. And then this is the methodology of augmentation we are going to use for our problem. Then we need to have the additional information. So, we need to have we need to determine additional information which we want to store in the data structure.

And which will help us to solve our query, which will help us to help us in our problem. So, our problem is the interval search problem. So, we have given time intervals and we have given a query intervals, and we have to find out the interval which is overlapping with this query intervals. So, that that is the problem. So, we need to determine the additional information which will help us to solve our problem. So, this is additional information we will keep the so, this is the key this is the interval basically. So, we will put this intervals over here and here will use the max m is the so, basically we keep this interval in a red black tree and using the key value as the left in point. So, red black tree. So, basically red black tree, but key is so, so we will keep the interval each interval is a node and the keyed key on low or the left endpoint. So, each interval is a node. Each interval is a node and the key value the key we are based on the key we are we make the binary search tree, based on the key value we are making the binary search tree if a node key value is less it should go to the left part if is more right part like this.

So, we will make this structure the key using the left endpoint or the low of this I. So, each interval is a node which is keyed on the low endpoint, and we are keeping the this field we are keeping the maximum value rooted at that subtree. This is the basically m of x m of x is the largest value largest value in the subtree rooted at x, largest value in the subtree rooted as x, x means interval x I into x, x is an interval x is an interval.

So, x is an interval. So, it has to endpoint low and high. So, based on the low endpoint, we make the tree the binary search tree not only binary search tree is the red black tree. And then this additional field we are keeping to store the maximum value of the of the maximum largest value which is stored in the interval rooted at x. So, we have the example we have given example. So, let us draw the tree. So, with this additional field.

(Refer Slide Time : 12:45)



So we just have this intervals 5 11 this are the low and high endpoints.

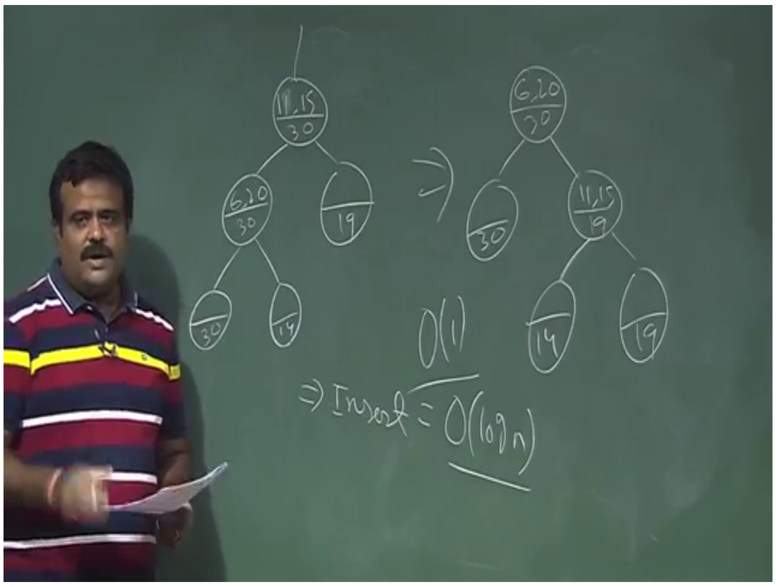
So, each interval is a node, and which is keyed as a low endpoint 8 9 sorry, 4 8 15 18 15 comma 18 and we have 7 comma 10 yeah. So, this is the these are the intervals we have given and this is the tree we construct and based on the low endpoint, if you see 5 is less than 17. So, 5 is here a 4 is less than 5 like this. So, 15, 15 is less than 17, but greater than 5 it is here. So, 7, 7 is less than 17 7 is greater than 5 or 7 is less than 15. So, 7 is the left. So, this is the binary search tree. Only binary search tree we can make it a red black tree by giving the color this we can put black, these 2 black, these 2 black, this is black and this is red and this is red.

So, we can put this nils. Now we need to so, this is the red black tree this is our underlined data structure red black tree. Now we have a additional bit of information which is basically max bit. So, this is the interval interval x and this is the max of x. So, max bit means the maximum. So, this is x. So, maximum. So, largest element rooted at that tree now what is the. So, this the nils. So, they are 0 basically what the largest element rooted at this tree 10 because the that is the high endpoint. This is 8 this is 23 now what is the largest. So, this is the 10 is the largest in the left subtree and there is nothing in the right subtree and this is the largest here.

So, this is basically 18. Now what is the largest over here? 8 is the largest in the left subtree, 18 is the largest in the right subtree, and the largest in this interval is 11. So, this is basically maximum of these 3. Similarly here 18 is the largest in the left subtree, 23 is the largest in the right subtree and the largest in this interval is 19. So, maximum of these 3 is 23. So, formula for m of x basically maximum of these 3, high of x high of x then maximum of left of x and maximum of right of x. So, this is the formula for this additional information ok.

So, this is a red black tree with this additional information. Now we need to maintain the how this additional information can be maintained when we will do the modifying operation like insertion and deletion. So, for that coloring will not be affect much. So, we need to look at this rotation operation, how it will affect in the rotation operation. So, to look at that let us just try to see whether we can maintain this additional information, with the same time while we are performing the rotation.

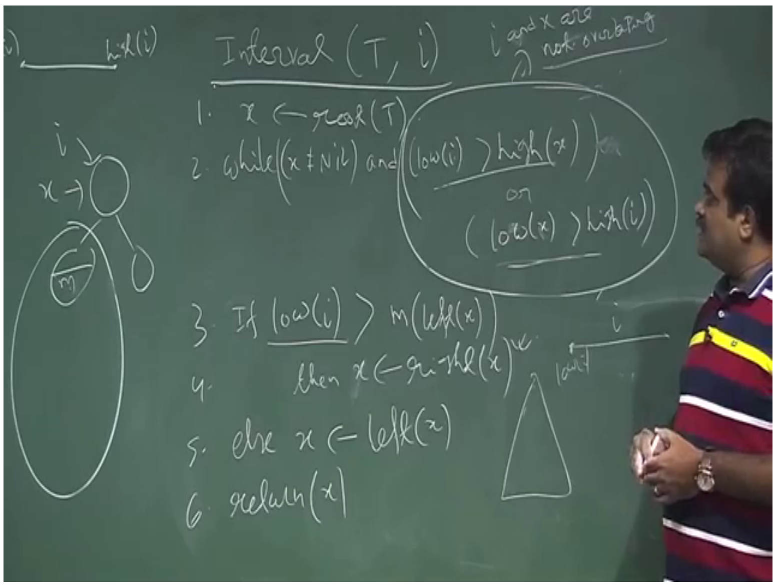
(Refer Slide Time : 16:47)



So, like this suppose we have this interval 11 15 6 20 and this is the say so we have a left subtree we have a right subtree over here. So, this is 30 say we have 30 over here. Now suppose we have we make this up this down this is alpha beta this is gamma, this is a subtree rooted at this are the subtree. So, this is a tree. So, this is part of the tree we want to see the how we can perform the rotation operation, and while performing rotation operation how we can fix this extra additional information in the same time. So, this is the right rotate on this. So, you want to make this up and this down. So, so 11 15 will come down and then this will be hanging here 30 and alpha beta. So, alpha this is beta this is gamma ok.

Now this will be maximum of this 3 19 and this will be maximum this 3 30. So, this can be. So, this can be fixed order of one time with the same timing with the rotation operation. Because rotation also need to change the point of view pointer. So, this is this is not a So that means, this imply insertion and deletion will take the same time as red black tree original red black tree insertion and deletion. So, that wise that that so this augmentation is, so now, we will see how this extra bit of information this augmentation will help us to have the search or interval query. So, that is the interval search ok.

(Refer Slide Time : 18:51)



```
Interval (T, i)
1. x ← root(T)
2. while (x ≠ Nil) and ((low(i) > high(x)) or (low(x) > high(i)))
3. If low(i) > m(left(x))
4.     then x ← right(x)
5. else x ← left(x)
6. return(x)
```

So, we have a set of intervals we have a set of intervals which we are maintaining in a that augmenting red black tree. And we search a query interval which is over we want to see which is overlapping with this interval. So, we have a tree over here which is based on our S this is this tree is coming from S set. So, what we are doing we are taking the root of this root of this tree. So, basically we have this tree which is basically the intervals like this like this So on. So, this is the this is our tree and this is the root, this is the on S set we are maintaining this thing. So now we have to so, given an interval I, I is basically we have given lower of I and we have given I of I. So, basically we need to find an overlapping interval with this. So, how we do so now, if there is element. So, until see if x is not null; that means, if there is only the root and, and this low of I if the low of I if the low of I is greater than high of x. So, this (Refer Time: 20:58) or low of x low of x is greater than high of I. So, we have this we have this interval. So, this is our x this is the root.

Now we check whether I is overlapping with this. So, this is the check whether I is not overlapping. So, how to check? So, we have a x interval. So, x is having here. So, we have a x interval. So, x is basically having 2 endpoints this is the low of x this is the high of x and we have an interval I, which is the query interval now when it will not overlap with this x, x is the root. Now we check whether it is overlapping with the root, if it is not overlapping then we will go for the left part or right part (Refer Time: 22:02) basically. So, before that we have to check whether it is overlapping with the x or not. So, how to check that? So, basically so, when it will not overlap, either I is completely this side. So, this is low of I and this is high of I; that means, if high of I is less than low of x this is either this or if it is completely that side. So, completely that side means if low of I is greater than high of x. So, either of these 2. So, this means the interval is completely that side and this means interval is completely this side. So, this means they are not overlapping, this means this condition means this whole condition means I is I and x is not overlapping, not overlapping not overlapping.

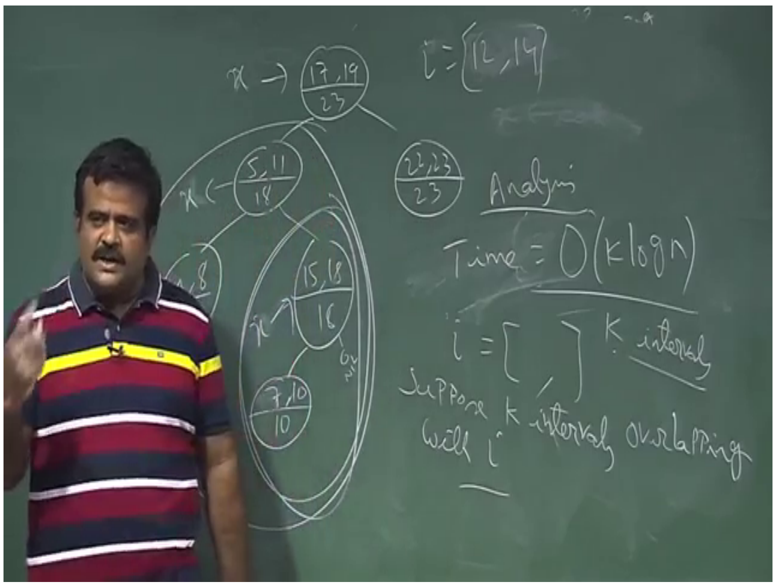
If there overlapping then we can return x, if this is false either it is reached to a null nil. So, so or it is overlapping if it is not overlapping, and it is not nil then we have to go further left or right part. So, this that overlapping then we have to go further. So, how to go for that? So, for that So, I and x is not overlapping by this condition then what we do? Then we do we just check if left of x is not null.

And low of I, if low of I is greater than max of sorry, if low of I is if left of x left of if low of I just a minute. So, how to check this? If low of I is greater than m of left of x; that means what? So, m of left of x so that means,. So, this is the m of left of x, So that means, if m of left of x is less than low of I So that means, there is nothing interesting in the left part. So, we are looking for a interval which is overlapping with I, if the low of I is greater than m of left.

If low of I is greater than maximum largest value rooted under this part then there will be no interval in the left part which is overlapping with I then we must go for the right part, then x is right of a x else x is left of x that is it. So, if either then we return finally, we returned x. So, either x is nil so that means, there is no interval overlapping with x or x is this. So, this means is this clear. So, this means if. So, if the low of I if the interval we are looking for. So, this is the low of I this is I ith interval. So, low of I.

If the interval we are looking for low value is greater than the left part of the x the maximum value of the left part of the x, then there is nobody interval will be there which can overlap with I. So, we have go for the right part of the I. So, that is the idea. So, this is the pseudo code for inter interval search and the what is the time complexity for this? So, time complexity is basically log n. So, we will do just quick example. So, we will just do some example on this. So, let us have the interval 17 19 22 23 5 comma 11, 4 comma 8, 15 comma 18 and we have 17 comma sorry, 7 comma 10.

(Refer Slide Time : 26:45)



And we have the nils and we have the color. Now suppose we search we search the I is equal to we want to search the interval 14 and 16. So, low of I is 14 high of is 16. So, we will just execute the interval search we have seen now. So, we start with the root x and we check whether this is overlapping with this. So, 14 16 is completely this side I means it is not overlapping. So that means, and it is not null also. So, we then we have to fill this part also. So, this basically 8, this is 10 this is basically 18, 18, 23, 23 ok. Now what we do we just take x to be root.

And this is the nor overlapping, now this is basically 14 is basically less than 18. So that means, we have to look at this part of the tree, if 14 is greater than 18 then there is nothing interesting over here. Then we have to go for the right, but here 14 is less than 18. So, now, our x is this. Now we check whether this is overlapping with this 14 this is not overlapping with this, now we check 14 left part of this now left part of this maximum value is 8, but 14 is greater than 8. So, there is benefit to going to the left. So, we will go to the right of the tree. Now this is our x. Now we compare this and this will be overlapping. So, it will return as this return 15 18 this interval.

So, 15 18 will returned. Now suppose we want to search for interval which is not there. So, suppose we want to search say 12 and 14. So, how will do? So, we will start with x over here. So, it is not overlapping. So, now, this for 12 is less than this. So, we will go to this part now this is our x again we check it is not overlapping. So, again we compare 12 with these 12 is greater than we have to come this part this our new root, and then we compare with 12 and this it is not overlapping, and we check 12 with 12 is greater than we go to the right is nil. So, basically we stop and it is returning us nil so that means, there is no interval which is overlapping with this. So, analysis of this so, the time is basically order of log n to report one interval ok.

But the question is if suppose there are k intervals is overlapping with this and we want to retrieve all of them. So, what then how we can do that? Suppose there are k intervals which are overlapping with this interval I, which the given interval I and we need to return all these k intervals. So, how to do that? Suppose k intervals overlapping with I with the given interval with the query interval I. And we want to return all the interval how we can do that?

So, what we can do we can first get the interval which is overlapping with this and then we delete it from this, augmented red black tree and that deletion can be done in logarithm time, again we make the search after deleting that it will give us the second interval which is overlapping with this. Again after getting that we delete it. So, basically the time will be k into log n. So, this is the time to report the k interval which are overlapping with this. Because every time once we got the interval which is overlapping, we will delete it because this is the red black tree deletion will be in log n time and after deleting that note that interval we again search we again perform this interval tree operations, searching.

And then again it will take logarithm time. So, this way it is the k log n algorithm, this is the output sensitive algorithm ok.

Thank you.