NPTEL
NPTEL ONLINE CERTIFICATION COURSE

Course Name
Fundamental Algorithms:
Design and Analysis

by
Prof. Sourav Mukhopadhyay
Department of Mathematics
IIT Kharagpur

Lecture 12: Augmenting Data Structure

Okay, so we will talk about augmentation of data structure, suppose augmentation means instead of having a data structure from the class we want to make use of some existing data structure and we want to add some extra information there that is called augmenting. And we want to take help of that data structure for our problems. So we will talk about two problems in this lecture.

(Refer Slide Time: 00:48)

One is dynamic order statistics that means OS select. So we know the select, we know the order statistics it is basically finding the i^{th} smallest element in a given input and there out input is this statistics, input is fixed, I mean then that is changing it, or we are not inserting or will be sending the input.

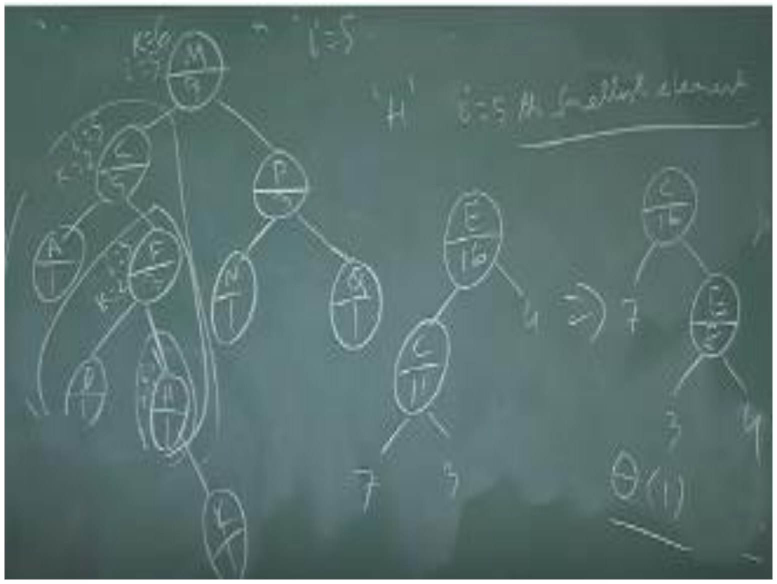
So but here S is dynamic set, S is dynamic set that means we are aligning the insertion insert and deletion. So we are aligning these two operation in S, so that way it is a dynamic set. So S is a dynamic set which is the, so now we want to perform this problem in this S, that means we want to have this which will give out the i^{th} smallest element in S, the set S which is having n element okay.

But this S here is dynamic set so it is any element can be insert in this or any element can be delete in S from S. So how we can maintain this, how we can have a data structure for this problem. So we will take help of some existing data structure then we will do some extra, we will put some extra information and then we will use that, so that is called augmenting.

So we will use the Red-Black tree as our underlined data structure. Because this is a balance tree, since this is balance so we want to use this as our underline data structure and we want to give more information for solving this problem. What is this, so this is the key value and we want to keep the size, size means the size of the tree routed at this node that is the size of the tree. So and this is the key value.

So this we want to, this is the augmenting we are using the existing Red-Black tree and there we are putting this extra bit of information there storing the size of the tree. Okay, so let us take an example.

(Refer Slide Time: 03:48)



Okay suppose this is our tree, these are the values key value C, P, A, F, let us make it little big so that we can have this other bit of information. So this is N, this is Q and we have another two nodes I want to make it little big this is D, this is H okay.

And we obviously have the Nils, so this is a red-black tree we can put the color on this tree and we have Nils over here, and the size of the nil is convention is, we put it 0 okay while we implement this. So now what is the size of this, so this is 1, 1, so each node is having two filled one is key and one is the size of the tree routed at this, so this is the size 1, 1 so size will be 1, size will be size of left sub tree, size of right sub tree plus 1, so this is 2.

So size will be 1 again here, so size of left sub tree, so this is the size of, size of this is basically, so this is the formula, size of a node is basically a size of the left of X last size of the right of X, last node X itself. So this is basically 3, 4 and sorry this 3, 1+ this is 3, so this is 5 and this is 1, this is 1, this is basically 1+1 so this is 3 and this is basically 5+3, 8+1=9 so this is the, our new data structure which is basically red, black tree, but we have extra function which we are storing the size of the trees rooted at that node, okay. So now if we have this data structure now how we can do this OS select so we want to okay so we want to so this is our tree so what we do we will just take that so this is the root of the tree so x is the root of the tree or we can just take T and this i so tree is formed OS select is, basically we first take the x as the root of the root the tree.

And then we defined as k is basically size of the left of x +1 so can you tell me what is this k represent k is basically size of the length so k value of this is what size of left of x +1 so this is 6 this we can check nothing but the rank of this node that means in a sorted order if we sort this element then the position of that node is basically rank that is the k because if you them in ordered reversal that will give us the sorted one so this will all the will between that so this is the size of the left sub tree 5+1.

So this is the basically, this is nothing but the rank of x now we are looking for heights smallest means we are looking for element whose rank is i so now the code is similar if i=k what we do we return the x we return x else if i< than k else if i< than k what we do we have to look at the left part of tree then we call, then we call the same OS select with the left of x with the i same i it is the same code as we discuss in our select okay now here if not else if i > that k else that means i > than k then what we have to do.

We have to go for the right part of the tree then we call then we return OS select right of the tree, right of x but with the i-k smallest element okay and we can have a checking whether yeah so we can just yeah so this is the i-k smallest element in the right part of the tree okay so this is the code now if we for this example suppose we want to find out the 5th smallest element suppose i = 5.

Suppose this is our input i=5 5th smallest element so this k is so i=5, now rank of this node this is the 6th smallest element so this is not so now i< than 6 so we have to go into the left part of the tree =, so we call this recursively on this sub tree left part of tree with the same value of i. So if you do that now again we calculate this so this is basically now we are looking for i=5 now if you consider only this tree now what so the rank, rank is basically 2 for this node in this tree sub tree.
So now i>thank 2 so we need to go for the right part of the sub tree with the i value i-k so 3 now we are looking for 3th smallest element in the sub tree now what is the rank of this rank of this is

again 2 so this is k sorry this is k rank of this element on this sub tree so now we now i> think we have to go for the right part of this tree with i=1 and the rank of this is k is 1so we will return this.

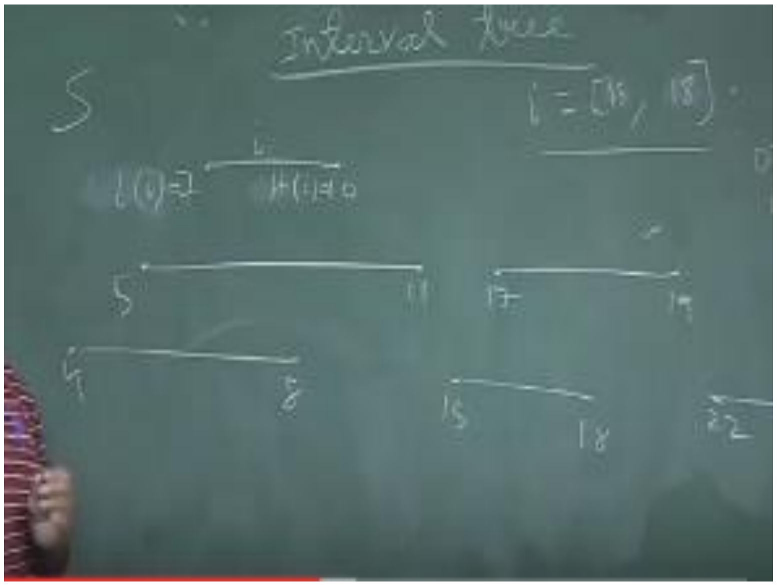
So this is the 5th 8th is the 5th smallest element, okay, so now this is the OS select now this side is dynamic so that means any node can be in joined at any point of time and nay node can be deleted so while joining the node we need to follow the inside algorithm for red, black tree insert, okay. So there we need to check we have this is a alimenting of the red, black tree so we have check how we can this change this extra of information while we are doing the modifying operation so that we need to verify whether this is easy or not so suppose we want to insert say here we want to insert a node which is k, so k is basically less than A we will go here k>c we go here k > A if we go here so we will put k over here. Now the same way so this is one now the same way we will just change this.

Now the change how much time it will pay into c while we do the modifying operation, so suppose this is the situation we have P and we have 16 over here and we have c we have 11 over here now we say this is the size now this is the size of the this now if we perform the right operation on C, the C will come up and E will come down, okay E will come down now C will be having this 7 and E will be having this C4.

Because this is α this is β C4 now how to change this values? Now this is basically this plus this plus 1 so this is 8 and this will be basically 7 + 8 +1 = 16 okay so this is also can be done in the constant time along with the changing with the pointers, so this is no problem here for this for maintaining this extra β information, okay. So this is the even the, for the re-coloring also we do not need to spend extra time on this, okay.

So this is the augmenting the data structure, okay. So now we will talk about this another problem which is called interval search okay.

(Refer Slide Time: 14:17)



So interval search it will basically will form the interval tree so the problem is we have given some interval time intervals say we have given say time intervals like 7, 10, 5, 11 say 17, 19 we have some given interval 4, 8 and 11 say 15, 18 and say 22, 23.

So these are the suppose this is the this is the interval this is if this is i, i denote this interval at the i this is low (i) so lower end point and this is the high (i) so we can just write h and h (i) l(i) and this is the h(i) so low end point high end point, okay. So now suppose our square interval is this say we want to I we have a given interval i say i say 9, 10 okay or say yeah 15, 17 15, 18 now we have to this is a given interval say.

These are the interval these are there reign intervals and we have a given interval i and which we need to find out what are find out a overlapping interval with this. So you need to find out overlapping interval with i, so who is overlapping? So this is 17 yeah so this is overlapping this is also one we can just report only one so this is the problem. So we have to report this interval 17, 19 and this set is dynamic set so we want to we have a set of intervals which is set S.

S set is basically set of intervals that are reign intervals and this set is dynamics so interval can join at any point of time any interval can get deleted at any point of time and the query is, we have given a interval query interval and we need to find out a interval an interval which is over latched with the query interval so that is the problem so for to solve this problem we will use a we need to have a data structure.

So basically we will do the other structure argument here also so we will use the red, black tree as our as our underlined data structure and we will do some we will add some extra bit of information there.

(Refer Slide Time: 17:30)

So basically we will use red black tree here also red black tree and by keyed the left end point or low, low or left end point as a key, okay. We have the intervals so when you get the tree we have to have e filed which is called key field.

So that we can use that value to make a this binary search tree that key value is the left end point and what we know, what we do the augmenting this is the interval and here m we put m is the maximum value in the, in the tree rooted at that node. So m is the, so this is m(x), m(x) is the largest value or maximum value in a tree rooted at, at x that is the m, okay. So let us draw the tree which we have for our this given interval, so 17, 19 then this is 5, 11 this is 22, 23 so this is 4,8, this are the interval we are given and we need to form a Red-Black tree 7,10, okay.

So now we need to so and we have the Nils over here, now we need to fill that another fields maximum element so how we different the m(x), m(x) is basically maximum among this three numbers. So this is basically high(x) high in point, high(x) or m(x) left of x or m(x) of right of x whichever is the maximum it will be the maximum. So this is basically 10, so if you have tree rooted at this 10 is the maximum in the tree rooted at this, and this one will be the 8, and what about this one, this one will be the maximum of left of maximum this, this and there is nobody over here so 18.

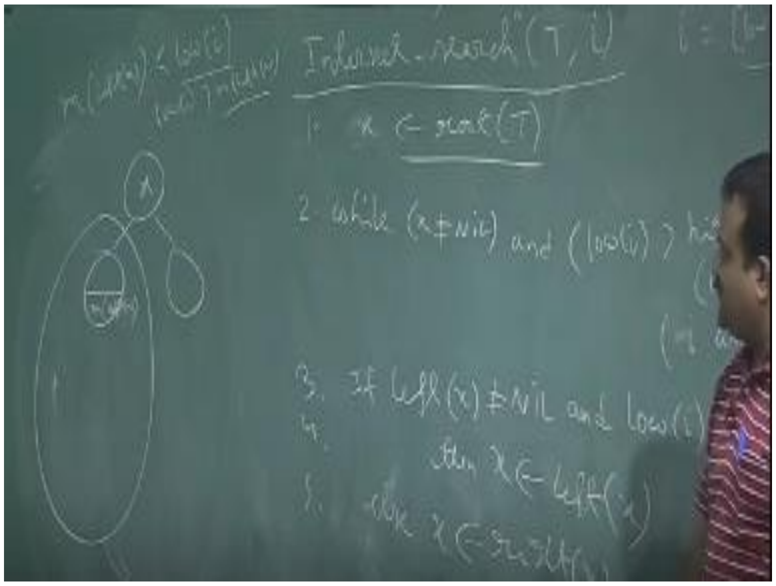
And this will be maximum between this, this and this, so this is 18 and this will be again 23 and this will be maximum between 18, 23 and this so this is basically 23, okay. Now we can just for a practice we can put the colors so that we can a Red-Black tree so which color we give so we can put this red and we can put this also there, now we can check this is a Red-Black tree, and the key we use this left in point as the key value, so we can just 17, 5 is so it is satisfying the binary search tree property.

So 5 is less than 17 you can just easily verify this, okay so this is our new derived structure not very new it is basically augmenting of the existing data structure Red-Black tree so now we want to see if we have this how so we are this a dynamic set so we are allowing the insertion and deletion so you want to see how much time we are spending for doing the modifying operation. If you want to insert a node then we have to change the color we will not take time we have to see the restructuring the rotation operation, so if we look at that suppose this is the interval 11, 15 and this is 6,20 and say this is, this is out α, this is our β suppose we have a 30 over here, we have a 14 over here so this will be 30 and say this is 19 this the γ.

Now if we want to do right rotate on this so this will come here, this will come down. So this will basically, so we have C over here, so 6, 11 20 will come up and this will be coming down and we have this will be hanging here 30 and, and two will be hanging here 14 and 19 and then this value is basically maximum of this, this is 19 and this is value will be maximum this, this, this.

So this is also a constant time, while we are changing the pointers we can do this, so this modifying operation will not take mush time so that sense we are safe. Now we have to so this we can very use, now we have to look at the, the search code, how we can if we have given interval, how we can have a overlapping interval, so that is the interval search and this tree is called interval tree.

(Refer Slide Time: 23:34)



Okay, so let us write the c2 code for interval search, suppose we are searching this interval i, okay so in here we are given a tree, we form we have set s n element and we form the tree this new tree which is basically a Red-Black tree with some more better information. And we this is the input so this is we are trying to find the i is an interval.

So i is having low of i high of i so this i is an input and we are trying to get the internal reaches overlapping with eyes so what we do we first take x as a Л of this tree now we check so x is an internal sorry x so x is an interval so x is having low and high low of x high of x now we have to check whether this interval is overlapping with x on nit if it is overlapping we will return x.

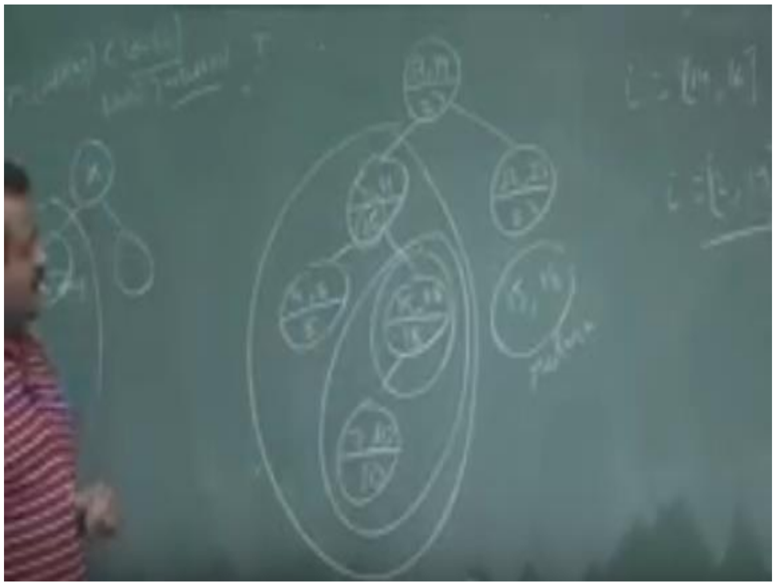
But if it is not how to check it has not a overlapping so if i is completely here i is completely here means if the high of x if less than low of x if the high of x is less than law or i is completely there that means if the high of x is less than low of x so that we have to check then there is no overlap okay so if while x is not nil and this is the checking for known overlapping and low of x low of i if it is better than i(x) and or low of this is x low(x) small x sorry low of (x) is better than high of i if this true either all of this.

Then that means not overlapping this is basically i and x at non overlapping do not overlap so the we have to go down so that means what we need to do down if they are not overlapping, and it is not an nil note then we have to check so this is basically go we have to go either left or right now if left of the (x) is not nil is not nil and low of i is less than aim of left of (x) then we go for the then x is = left (x)else x = right (x) okay so that mean so suppose we are looking at so this is our x now this is our aim of left (x) now if aim if a left(x) if this is if it low of x is less than this then we will go for left it low of x is better than this.

So suppose aim of this is less than low of (i) so that means we are looking for a looking for a overlapping while low of i is better than this that means low of i is better than aim left (x) so that means the substitute at is it maximum halo is less than low of i then there is no position of going to the left part then we are going to the right part so that is discuss else case if the maximum halo in the low (x) is less than left of x is less than i then we will go to the right but because there is no question there is no overlapping because the interval itself is better than this so we will go to the right part of it.

So this is the code so what is the time complexity of this code it is basically log n some of let us taken example week example how it is working.

(Refer Slide Time: 28:50)



Okay so the same example same tree suppose we have this tree 17, 19 i from 11 22, 23 4, 8 15, 18 7, 10 okay now if feel this 10, 8, 18, 18, 23, 23 suppose our query interval is 14, 16 suppose we are trying to find out the overlap interval with the 14, 16 so we is there any overlap with this no then we go to the we check this, this with the low of this if the low of so this is so then we go for this tree left tree because low of this is less than maximum halo so we should have something over here.

Now this is our x now we go and look at the left part of this, this is 8 now the our interval is starting from 18, so 8 is the maximum in this sub tree so there is no question of going there so we will look at there so we will look at there we compare with this so this is has no yeah this is having overlap so 14 15 and 18 so this is having overlap so we will return this one so we will return 15 and 18 so this is we return okay.

Now similarly you can check with the i=12 and 14 which is having no overlap so it will raise to the nil note and then we end there so this is the code so this is the has time complexity log n because we have to maximum we have to we are traverse the tree of Л to the leap sp the height is balance tree so the height is log n so the time is log n, thank you.