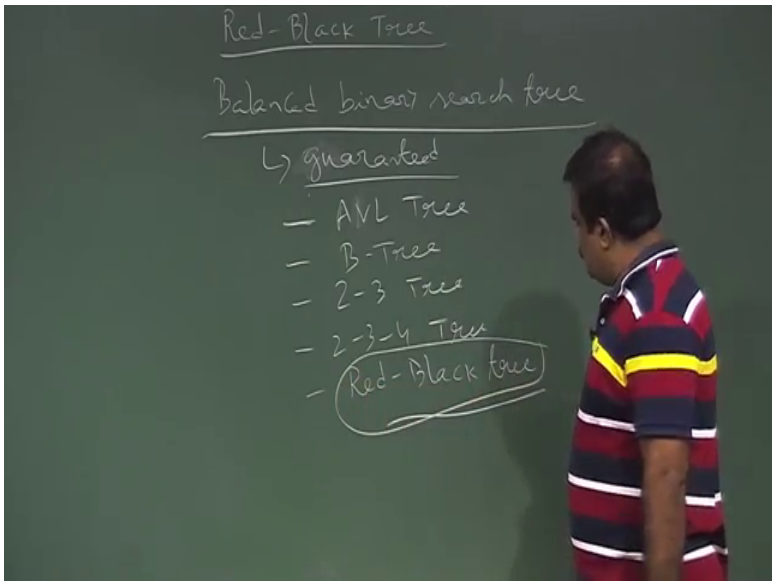
An Introduction to Algorithms
Prof. Sourav Mukhopadhyay
Department of Mathematics
Indian Institute of Technology, Kharagpur

Lecture - 27
Red Black Tree

So we talk about balanced search tree. So, we have seen the if we have a input if we have n element and if we can do a random permutation before forming the tree, then after doing the random permutation if you from the tree then that tree will be expected to be balanced, but there is guarantee because that is the average case analysis, but the worst case it may be quadratic it may not be balanced in the worst case.

So, now we want a guaranteed balanced binary tree. So, that is our class. So, red black tree one the example.

(Refer Slide Time: 00:58)



Red-Black Tree

Balanced binary search tree

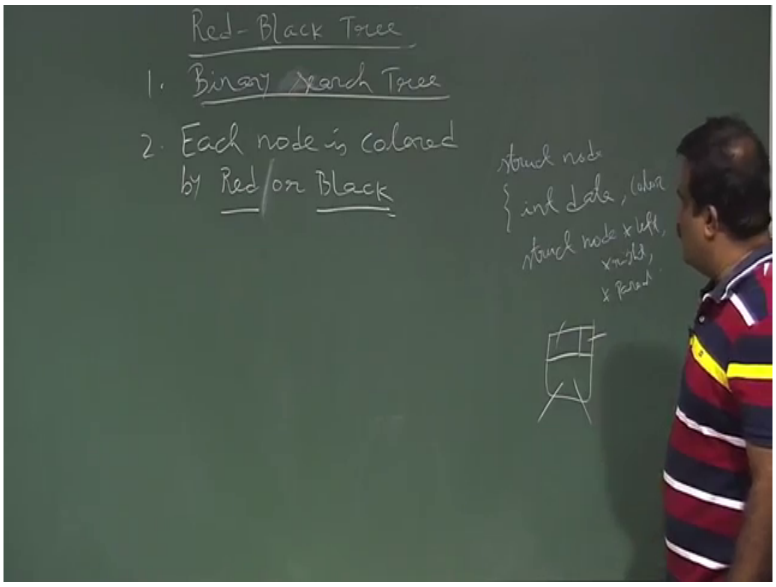
↳ guaranteed

- AVL Tree
- B-Tree
- 2-3 Tree
- 2-3-4 Tree
- Red-Black tree

So, we want balanced search tree binary search tree and this is guaranteed no randomly build nothing. So, we want guaranteed binary search tree because we want the balanced tree that is a good tree because we can perform the query in a logarithm time if we have the balanced tree if we can store our data in a tree which is balanced then we can just do the query in a logarithm time or many things we can do in log. So, there few example of a balancing and this is the guaranteed.

This is the no randomization this is guaranteed. So, we need guarantee we need the balanced binary search tree. So, there are some example of balanced binary search tree like AVL tree B tree then 2 3 4 tree 2 3 tree 2 3 4 tree and same red black tree. So, these are balanced binary search tree. So, we will talk about this red black tree in more details. So, what is the red black tree? So, what is the definition? So, red black tree is a binary search tree with some properties what are those properties.

(Refer Slide Time: 02:49)

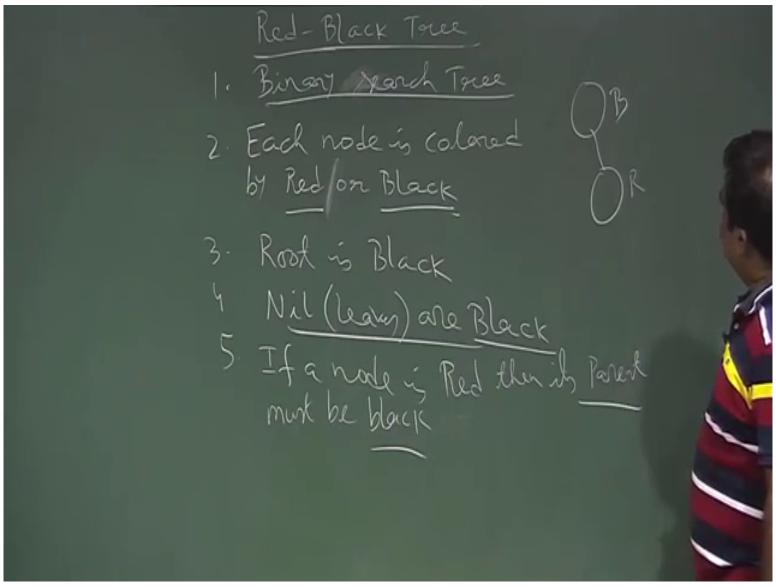


So, first of all it is a binary search tree.

So, binary search tree means we have that property. So, if you take any node in the tree x the value of this key value of this must be the all the left subtree has value less than x all the right subtree has value greater than x. So, this is binary search tree property. So, this is a binary search tree, that is the first when that is the. So, now, each node is coloured and coloured by 2 coloured, coloured by either red or black red or black. So, we give the coloured to each node. So, and we can only use the red colour or black colour. So, for that if you want to implement this. So, usually tree for tree we have we define the node like this struck node. So, in data

And then we have the pointer for pointer for the left child and may be the right child and maybe we need to have to have a pointer for the parent. So, now, we have a coloured bit also. So, use on one bit this is the data we use one bit as a colour either red colour or black colour. So, this is if it is one we can use it that is the convention we can fix by ourself, but each node has to be coloured by 2 colour either red or black. So, this is one property another property is roots are black.

(Refer Slide Time: 04:53)



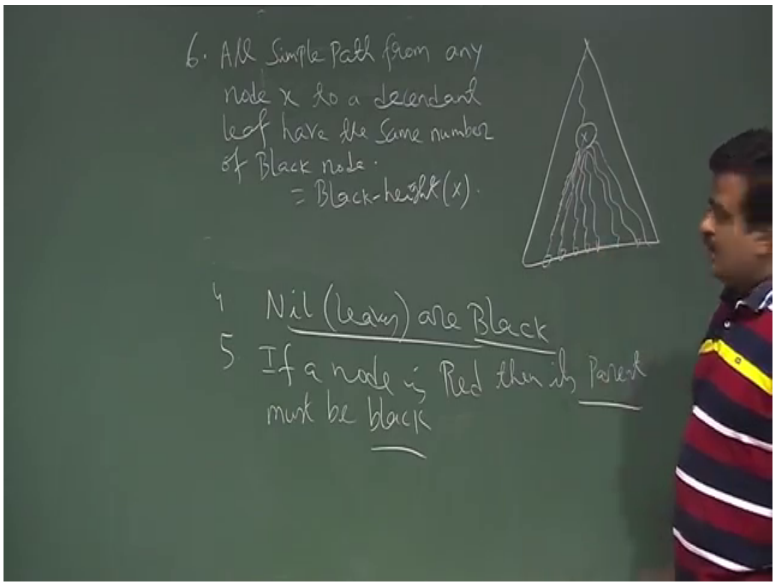
Red-Black Tree

1. Binary Search Tree
2. Each node is colored by Red/or Black
3. Root is Black
4 Nil (leaves) are Black
5. If a node is Red then its Parent must be black

Root is black and nil leaves nil are basically new leaf will come to that what are this nil leaves are black. Nil are basically the leaves, leaves means not may not be are not originally. So, we have a tree we have the leaves. Now we introduce some new leafs which are basically nil to make the tree to be complete binary tree because a node may have only one child. So, to make it complete we need to introduce this nodes nil anyways we will come to an example. So, nils are black and the next property is if a node is red then the parent then its parent must be black.

So, if a node is red if this is the red node, then the parent has to be black the parent cannot be red; so this one property. So, if an node is red then the parent has to be black. So, this is and the last and there is another property which is very important property which is give us the black height. So, this is the next property.

(Refer Slide Time: 06:40)


6. All Simple Path from any node x to a decendant leaf have the Same number of Black node.
= Black-height(x).
x
4 Nil (leaves) are Black
5 If a node is Red then its Parent must be black

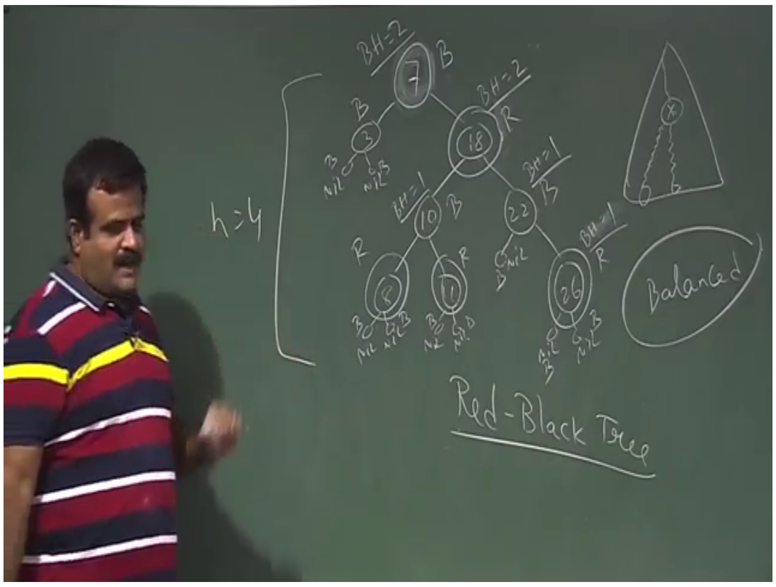
So, it is telling basically. So, if you take a node. So, suppose this is our tree, now if you take a node from this node if you just visit from this node to the leaves. if you just visit all the path from this node to the leaves so, simple path that we are not considering the loop.

Now for each part if we count the number of black node. So, this is a path from this node to the leaf this is another path from this loop to the leaf it is (Refer Time: 07:18). So, if you just count number of black node in the path. So, each path should contain same number of black node and that number is refer as a black height of this node. So, let us write this property all simple path from any node x to is to a.

Descendant leaf has same number of black node have the same number of black node and that number is referred as a black height of x. So, if you take any node in the tree and if you consider a path from this node to a leaf, and we count the number of black node if and then we take another path from this node to a leaf, and the same number of the number of black node must be same if there is four black node in this path, there has to be four black node in this path there has to be four black node in the path and that why it is the unique.

So, this is called black height of that node. So, this is just a number of black node from a number of black node from a path, in the path from that node to its to the leaf is called black height of this node. So, if the 6 property satisfied, then we call the tree is a red black tree and we will see because of this properties this tree will be a balanced tree guaranteed. There is no randomly permutation nothing this is guaranteed this is guaranteed, this is a this if our three is satisfying this property then the height will be log n that we will prove (Refer Time: 09:46). Let us take an example of a red black tree let us give an example of a red black tree, this is the definition of red black tree these are the property our tree must satisfied. Then the tree is called a red black tree.

(Refer Slide Time: 10:08)



So, suppose this is our nodes here 7 say 3 then 18, 10, 22, 8, 11 and we have 26. Suppose this is our number is given these are the numbers given and this is a binary search tree if you look at any node 10 18. So, if you look at 18 all the subtree rooted here are less than eighteen also subtree rooted here greater than 18. So, this is a binary search tree. Now we want to make it a complete tree. So, we will introduce nil nil are basically new leaves.

So, we will put nils to because this is having one node. So, we introduce the nil. So, from each leaf node or form each node which is having less than I means which is having one child or no child we will put the nil and these our new leafs. So, this we nils we introduced. So, now, this is the nils and now we need to give the colour of this tree now we give this root to be black, because that is one of the property root is black. So, this is. So, this is 7. So, root is black and these nil are also black this is one of the property we have nil are black ok.

Now we need to give the colour of this nodes so, that we must satisfy that six properties i mean those properties. So, which colour we give to this node. So, we can give to this node black and we can give to this node red, and maybe this node red, this node red this is black this is also black and this is red. This is we can try we just suppose we assign this colour each node now this is a tree binary search tree and each node is coloured and each node is coloured by red or black. Now root is black leaves are black. So, that is the one of the properties now if a node is red we know parent has to be black. So, this is red node.

It is parent is black this is a red node parent is black this is a red node parent is black. So, that is that property is also satisfying if a node is red that the parent are black. So, now, we have to see the final property which is called our black height. So, if we look at all the nodes. So, if you look at this nil, nil black height is basically zero now if you look at this node. So, this node what is the black height. So, if you look at this node so this node if you form this node to a leaf how many black node only one this node to a leaf. So, black height of this node is basically BH. Black height of this node is one now what is the black height of this node; if you take this path. So, there is one if you. So, that black height in the black height we must exclude the colour of this x.

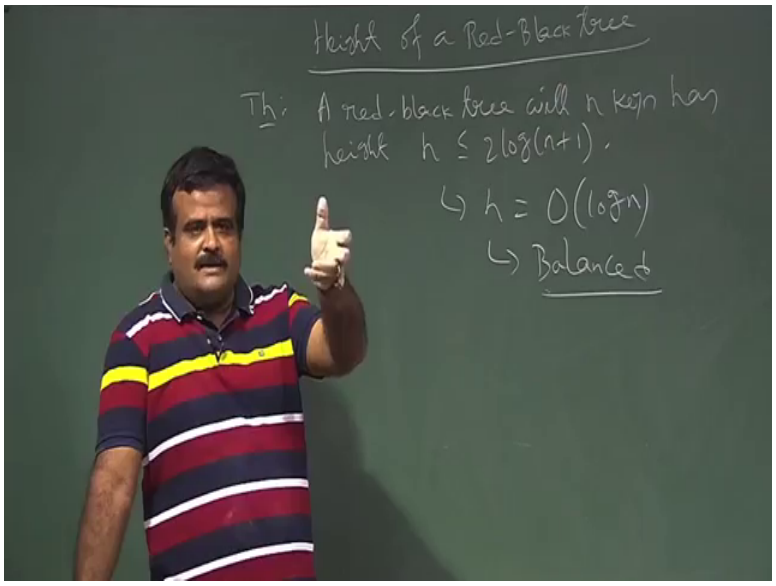
So, let us just recall black height is basically we take x and we consider all the nodes black nodes in this path and if excluding the colour of x if the colour of x is black we are not counting that in because excluding the colour of x. So, it is that that has to be noted we are not counting the colour of x if the colour is black we are not adding that we are excluding the colour of x. So, that is; that means, black height of this is basically also one because it has if you check it now black height of this node. Now this consider this path from this to this how many two black node 1 2. Consider this path this to this 1 2 consider this path 1 2, consider this path 1 2, consider this path 1 2.

Consider this path consider. So, from this node if we look at all the path simple path from this node to the leaf all are having same number of black node and which is basically two. So, the black height of this node is basically 2 and this black height of this node is now what? Now black now black height of this node is also consider this path it has one considered this path this is one considered this is point because we are not counting the black of this because that is excluded. Now what is the black height of this node? So, if you take a path from this. So, this is one two one two if you take path; so from this node to this leaf if you take the path.

So, we are not counting this black; so 1 2. So, just we can just check the black height of this node is basically 2. So, this is the red black tree because it is satisfying the all the properties we have listed. So, every node is coloured this is a binary search tree every node is coloured by either red or black, and if a node is and the roots and nil are black and if an node is red then the parent is black and the from a given node from a node x if we consider the path from x to a leaf, and all the paths all such path having same number of black node excluding the colour of x and that is called black height. So, we have the that that properties also satisfy. So, this is a could nil this is a black tree.

And this is balanced what is the height of this tree height of this tree is 4 and how many elements are there? 1,2, 3, 4, 5, 6, 7, 8. So, but this is a eight we in general we have to prove that this is the balanced tree. So, this tree is balanced, but we need to prove why this is balanced for n nodes, but this is one example of a red black tree now let us prove why this tree is balanced by a theorem. So, this theorem is telling. So, height of red black tree we will come to this example again and we will prove this theorem. So, let us talk about height of a red black tree.

(Refer Slide Time: 17:59)


Height of a Red-Black tree

Th: A red-black tree with n keys has height $h \leq 2\log(n+1)$.

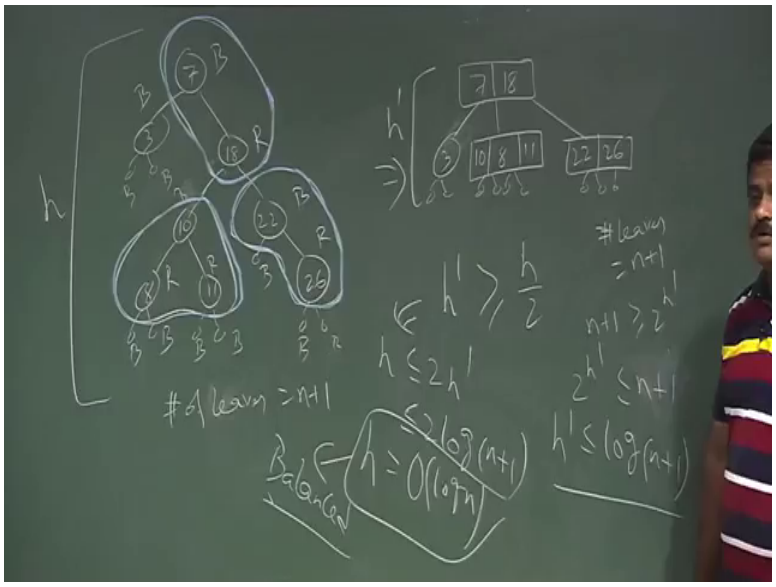
$\hookrightarrow h = O(\log n)$

$\hookrightarrow$ Balanced

Let us establish that this is a balanced tree height is log n height of a. So, this is we are proving by this theorem. So, this theorem is telling a red black tree with n nodes n keys has height h which is less than log of 2 log n plus 1. So, this is the theorem and this is telling us height is. So, this means height is order of log n.

So, this telling us balanced. So, red black tree balanced if you can prove this. So, how to prove this? So, to prove this we will use some intuition, intuition is we want to remove the red nodes. Now the question is how we can remove a red node from the tree. So, to remove the red nodes we know if a node is red then the parent is black. So, basically what we can do we can merge the red node to its parent because is black. So, that is the one way you can remove that red node. So, that we are going to do. So, how we can do that? So, let us just take that same example. So, to prof this will use that same example and we will see how we can remove the red nodes. So, basically we merge the red node use its parent because we know if a node is red the parent is black.

(Refer Slide Time: 20:06)



So, let us draw in side. So, 7, 3, 18, 10, 22, 8, 11 we have 26 and then we have the nil and the nil are black and this is red this is black this is also black nil are black and this is basically red and these two are red this is red this is black this is black. So, this was our example of a red black tree.

Now we want to remove the red node. So, to remove what we do we know if node is red its parent is black. So, we merge a red node with its parent because we know parent is black. So, if you do that this is our original tree, if you do that then what will be the situation. So, this is red. So, this we are going to merge with this, this is red. So, this we are going to merge with the parent and this two is red. So, this we are going to merge with the parent. So, this is the merging we are going to do so.

So, this is this, this is red node this we are going to merge with the parent every red node is merges with the parent. Now if you do that then we will have this tree this tree combined into this tree. So, this 7 and 18 is merge and it has one child three and we have another child which is basically having 3 nodes because this three node has merge. So, ten eight eleven and we have another 22, 26 and then the nil are there. So, this will having 4 nils and this will have 3 nils like this.

So, this tree will convert into this tree and this is nothing, but 2, 3, 4 tree why. So, what is the number of leaves in this tree for each node? So, number of leaf is either 2. So, the number of leaves is either 2 or 3 or 4. So, that is the reason it is called 2 3 4 tree and these tree is balanced with this tree is very nice structure. So, this tree is. So, this tree is convert into this tree. Now we want to talk about height of this two tree. So, what is the height of this tree suppose height of this tree is h and height of this tree is say h prime. So, suppose height of this tree is h and height of this tree is h prime. Now, what we can say about relationship between h and h prime.

So, can you say that can you say h prime is greater than equal to h by 2 because. So, we are merging the red node. So, if we consider a path which is giving us the height. So, if you consider a path now when it will be equal, when there are it is merging with half. So, when there are equal numbers in that path when there are equal number of red node and black node then it will be equal, but we know if a node is red then the parent has to be black. So, if you consider any path then the number of red node will be less than the number of black node because that is one of the property if a node is red then the parent has to be black. So, number of red node is less than the number of black node.

So, for these to be less than we need to have one red node more which is not possible because consider a path if a node is red than the parent has to be black. So, in any path the number of red node will be less than number of black node. So, that is the reason this height is the new height will be greater than h by 2 and it will be equal when the exactly same number of black and red node is there in the tree. So, this is the relationship we have. So, this is the height of this tree this is the height of this tree. So, now, what is the number of leaves over here in this tree? So, number of leaf is basically n plus 1 and this is same as here also because we are not merging the nil because nil are black.

So, number of leaf will be same number of leafs will be same n plus 1. Now this is having each node is having either to child or 3 child or 4 child; now two is the minimum number of child. So, from here we can say that n plus one is greater than two to the power h prime because if every node has 2 to the power if every node has 2 child then two to the power h prime is the minimum number of leaf it should have. So, n plus one must be greater than 2 to the power h prime. So, 2 to the power h or n plus one is less than four to the power h prime, but we do not need this will use this one. So, basically from here what we can say h is less than 2 h prime and which is less than equal to.

So, 2 to the power h prime so; that means, two to the power sorry it is just. So, from here 2 to the power h prime is less than equal to n plus 1. So, h prime is less than equal to log of n plus 1. So, these we are going to use here. So, h is from here what we can say h is less than equal to 2 h prime, which is basically less than equal to 2 log of n plus 1. So, a height of this tree is basically log n; so balanced. So, a red black tree bal balanced. So, this is guaranteed there is no randomness, nothing. So, this is guaranteed balanced tree. So, this is good news. So, if we can have such a tree balance tree then we can perform our query search in a height time and height is log n. So, the search or any other query can be handling log n time.

So, in the next class we will talk about the modifying operation how we can insert. So, this is a dynamic set how we can insert a element in a red black tree. So, modifying operation that we will discussed in the next class.

Thank you.