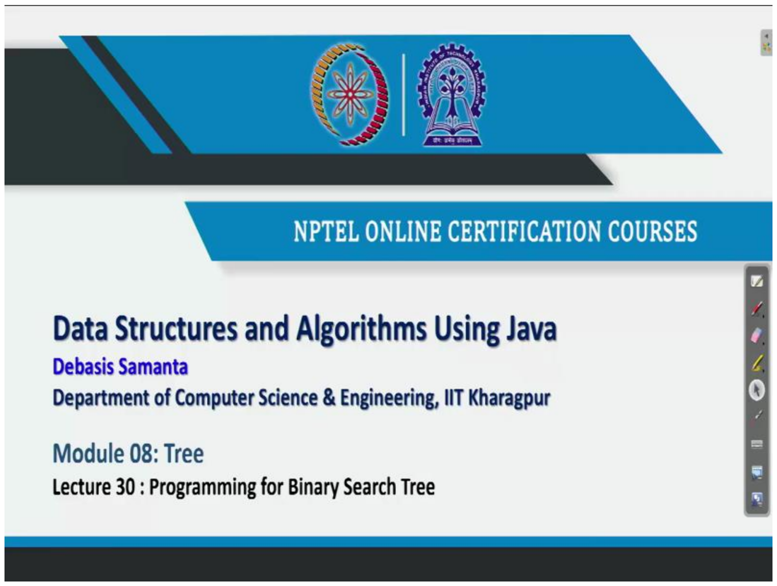
Data Structures And Algorithms Using Java

Professor Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 30
Programming for Binary Search Tree

(Refer Slide Time: 0:28)



NPTEL ONLINE CERTIFICATION COURSES

Data Structures and Algorithms Using Java

Debasis Samanta

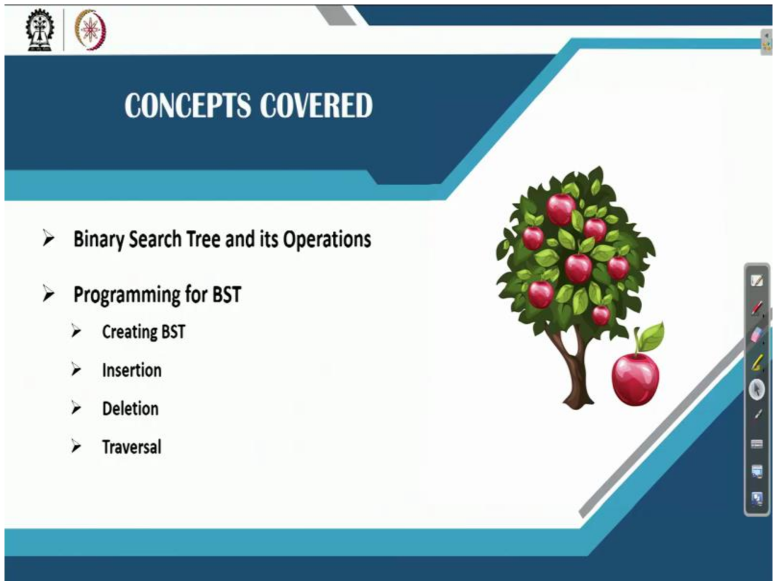
Department of Computer Science & Engineering, IIT Kharagpur

Module 08: Tree

Lecture 30 : Programming for Binary Search Tree

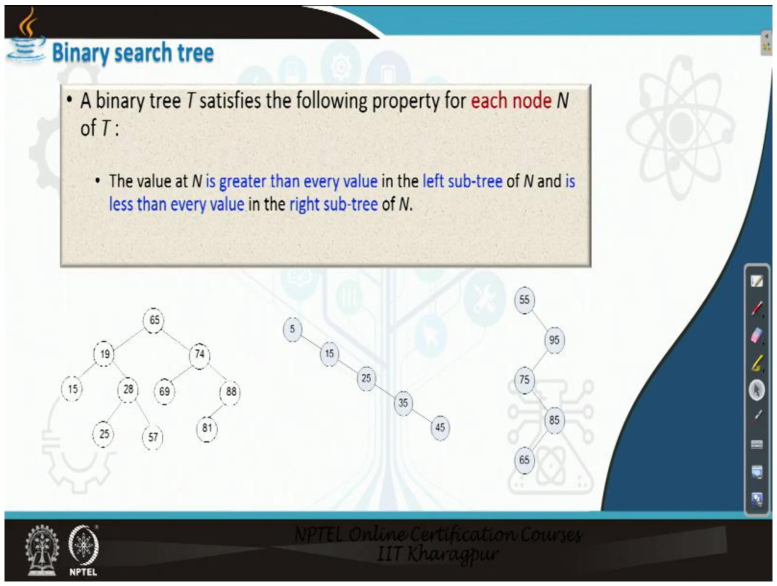
We have learned about Binary Search Tree. Now let us see, how the different operations including how a Binary Search Tree can be stored in a memory, and then programming for that. Let us learn about it. So in this video lectures, we will go for little bit programming aspects of, programming aspects with Binary Search Tree actually and Java programming.

(Refer Slide Time: 0:51)



CONCEPTS COVERED

- Binary Search Tree and its Operations
- Programming for BST
  - Creating BST
  - Insertion
  - Deletion
  - Traversal



Binary search tree

- A binary tree T satisfies the following property for each node N of T:
  - The value at N is greater than every value in the left sub-tree of N and is less than every value in the right sub-tree of N.

And here we will see exactly how fast a Binary Search Tree can be created, and then how the different operations namely insertion, deletion and traversals can be carried out. So, these are the basically task. And, okay fine, we know that okay, Binary Search Tree follow a specific property that we have already idea about it.

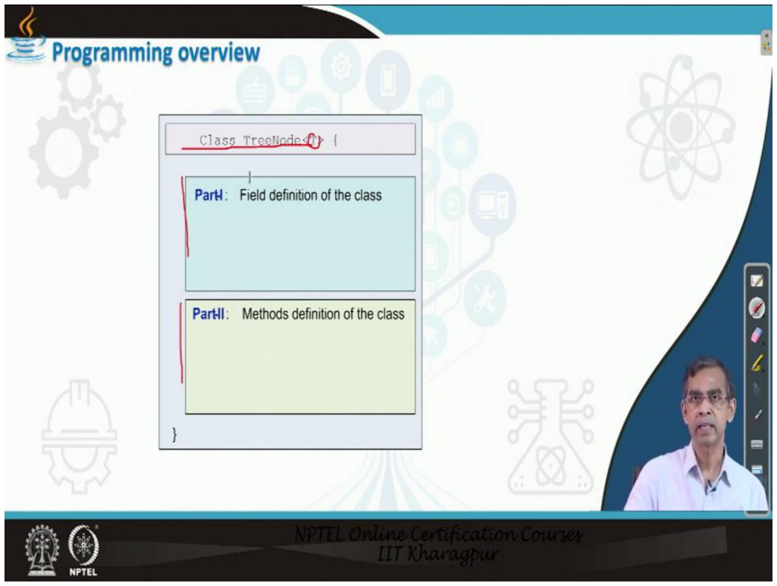
(Refer Slide Time: 1:18)

Programming for Binary Search Tree

NPTEL Online Certification Courses
IIT Kharagpur

Now let us come to the programming aspects of the Binary Search Tree. Now so far this programming is concerned, we will consider again generic programming. Generic programming means, we can create a Binary search Tree, or rather we can create a program in such a way that this program allowed to store Binary Search Tree of any type of data. So that is why the generic class needs to be defined here. Now the structure of the programming, it will be very similar to the other structure that we have already considered for link list, stack and queue like.

(Refer Slide Time: 1:53)



Programming overview

```
Class TreeNode<T> {
    Part-I:  Field definition of the class

    Part-II: Methods definition of the class
}
```

NPTEL Online Certification Courses
IIT Kharagpur

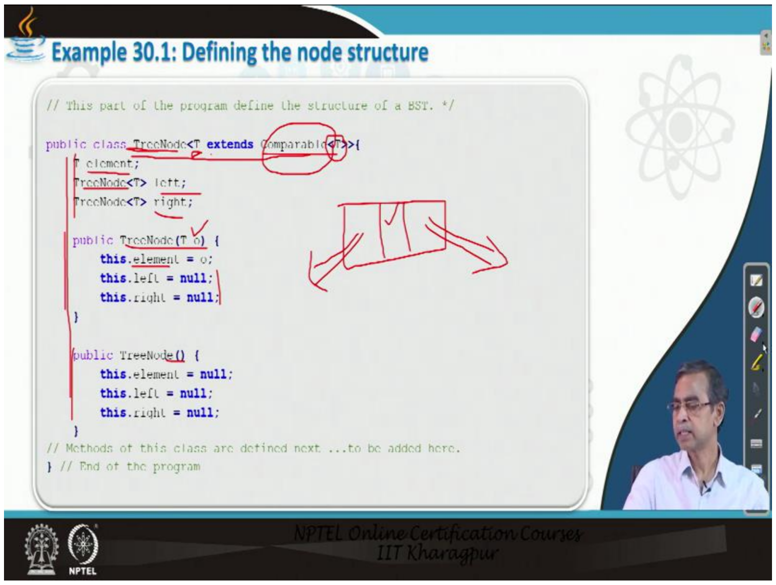
So, a generic class needs to be defined. Now, here I have given idea about overview of the structure. So that this is the declaration of a generic class. And it basically type that it will store, the elements in it, and then it should declare the different fields that is require, maybe lane, maybe link field, left child, right child, pointer and everything, and then their different operations. So different operations, actually the operations means insertion, deletion, merging, or searching, whatever the operations are there.

So all these operations are to be added as a members into this class. So these are the methods we can say. So the, our task is basically, how we can create a note, and then how we can define a note rather. How we can define a note of a Binary search Tree, and then that how a Binary Search Tree can be created. In order to that, we have to define a constructor. So constructor is another method, and then different fields like this. Now let us come to the discussion, it tells about the programming aspects, so that we can understand about it.

(Refer Slide Time: 3:04)

Defining Structure of Binary Search Tree

NPTEL Online Certification Courses
IIT Kharagpur



Example 30.1: Defining the node structure

```java
// This part of the program define the structure of a BST. */

public class TreeNode<T extends Comparable<T>>{
    T element;
    TreeNode<T> left;
    TreeNode<T> right;

    public TreeNode(T o) {
        this.element = o;
        this.left = null;
        this.right = null;
    }

    public TreeNode() {
        this.element = null;
        this.left = null;
        this.right = null;
    }
// Methods of this class are defined next ...to be added here.
} // End of the program
```

So first of all, we should define the structure of a Binary Search Tree, that means generic class. And this example can be taken a little bit carefully, so that we can understand about it. So this is a program which defines, the class definition of for a Binary Search Tree. So first of all, we have to define the nodes structure. Actually a Binary Search Tree is a collection of nodes. So that is why defining one node, that will be the starting node or root node.

And automatically we can adding node into it, inserting. So this way the Binary Search will grow. Now here you just little bit check about we are declaring a class. The name of the class is T note and here the type of the T is a generic, so it is a template. Now it is, here you see this class or this T extends comparable T, where T is any type. Like okay, actually what is the necessity of the comparable is that, that we suppose want to store some elements other than number or string.

Number and string by default, they are comparable but other type, for example, say student, book or some other type like, they are not comparable. So a any numeric data, including integer, long floats, whatever it is, they are comparable, string comparable. But other than integer string, no other data is comparable. So, we have to just extend comparable, in the sense that it is applicable to those comparable class or not.

So that is why, we limit the applicability of this one. So this means that all this T should be the type, which is basically, is sub class of the or class of the comparable class actually. So mainly this Binary Tree will be defined for all numeric as well as string like. For other type, it will not be applicable. For this things, if you want to apply, then we have to define this comparable first and then extend this node as a comparable class actually, then only it is there. So these are, this kind of example we have already exercised while we are discussing about Q structure you can recall.

Anyway, our next part is basically, defining the fields. First of all the element itself is the one part. Now, here you can see a node structure is like this. So there is a one is called the data part, then left child and then right child. It is the concept it is there. So here, this is basically element. This is the data part actually. Then this basically left, here is the left link, that means it stores the address of other node, that is why we make this is a type of T node, because it will store the type of left node, that means it will store the address of another node.

And right is basically this one, it will store the address of another node. That means is basically store the address of the root of the right safety, root of the left safety, whatever it is. Now here is a constructor, you can see, this constructor is defined. If we add one element O into it, then we can call this O, so initially O can be null. If it is not null, so initially element, this element O, means root will be O. So this constructor allow you to, I mean, create a tree if you pass already existing one tree, Binary Search Tree like. So O is maybe another Binary search Tree you can say.

And its left and right for the initially made null, that means it does not have any left child, right child like this one, if it is there. So anyway, another constructor also is a default constructor look like this. We do not pass any arguments, so simply we are null like. So this completes the declaration of constructors as well as field in it. Our next part is to declare the different methods those are relevant for this type of data structure. The relevant operations are basically insertion, deletion and maybe searching, whatever it is there. Now let us see how the different methods can be added one by one in this class declaration, to make the class complete and ready for use in any master program.

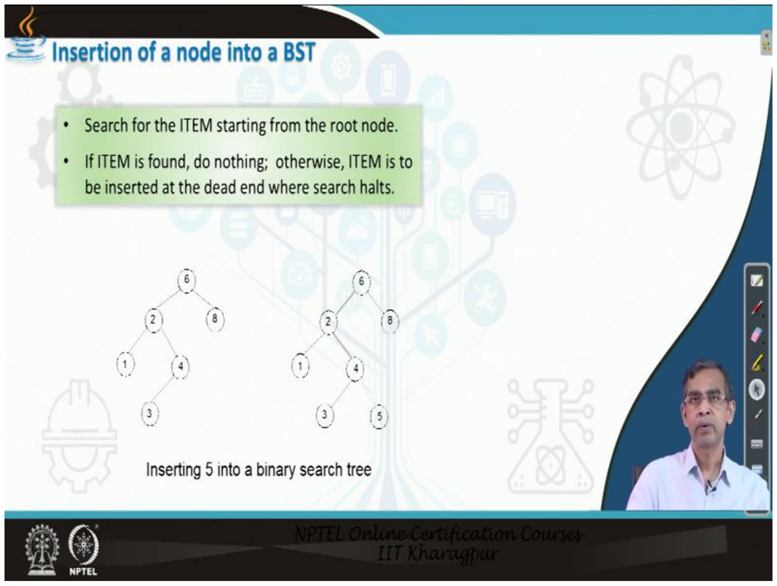
(Refer Slide Time: 7:21)

Defining Methods of Binary Search Tree

NPTEL Online Certification Courses
IIT Kharagpur

Method for Insertion

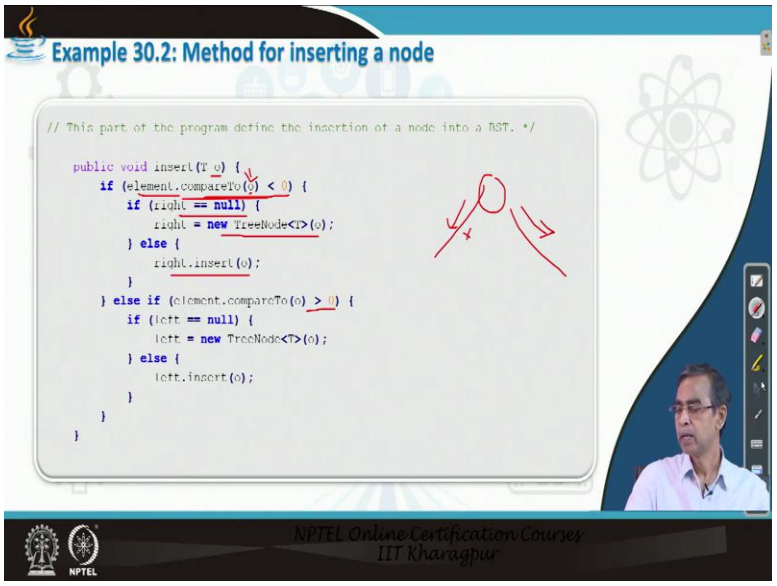
NPTEL Online Certification Courses
IIT Kharagpur



Insertion of a node into a BST

- Search for the ITEM starting from the root node.
- If ITEM is found, do nothing; otherwise, ITEM is to be inserted at the dead end where search halts.

Inserting 5 into a binary search tree



Example 30.2: Method for inserting a node

```
// This part of the program define the insertion of a node into a BST. */

    public void insert(T o) {
        if (element.compareTo(o) < 0) {
            if (right == null) {
                right = new TreeNode<T>(o);
            } else {
                right.insert(o);
            }
        } else if (element.compareTo(o) > 0) {
            if (left == null) {
                left = new TreeNode<T>(o);
            } else {
                left.insert(o);
            }
        }
    }
```

So now let us define the different methods of Binary search Tree. And the first methods, first category method is insertion. Now, you can understand, insertion we have discussed about. For the insertion, we have to check whether element exists or not. If it does not exist, then we insert it as a dead end like. So it is a example like. Now here is a program that you can follow, is basically implements the algorithm that we have given once and it is a Java code actually implementing that algorithm here.

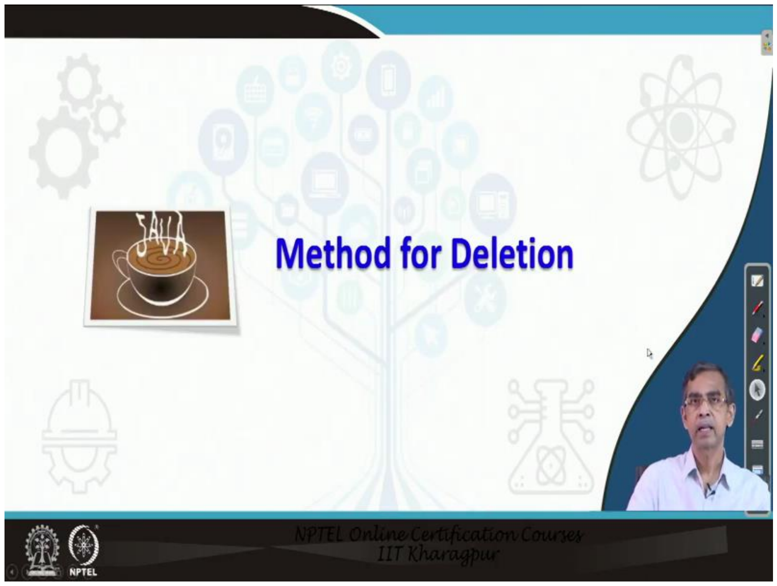
So this algorithm is basically if we want to insert the element O, then is basically first compared to, now here, compared to means element can be any type like, so compared to methods should be called for the objective elements. Elements is basically the value that is stored there as a data like, you can say. Now that compared to written minus 1 or plus 1 like. Minus 1 means, it does not match.

Plus 1 means, it is basically, minus 1 means less than, greater than 0. That is plus 1 means it is greater than. And if it is equal to 0 means it matches. So that is why compared to method will be written like this. So if we call for the current node, this compared to method passing O as an argument, that means it will compare the value of O with the current value in the current node.

And it according to the rule, if it is less than 0, then what you can do is that, then you have to check that if right part is null. That means it is a dead end or not. If it is there, then we just insert the new node at the right, what is called the child of that node. Otherwise, we again repeat the same procedure to go to the starting from the O node, I mean the node to the right part only. Now, on the other hand, so this basically it is deciding whether we have to follow this path, starting from this node or this path.

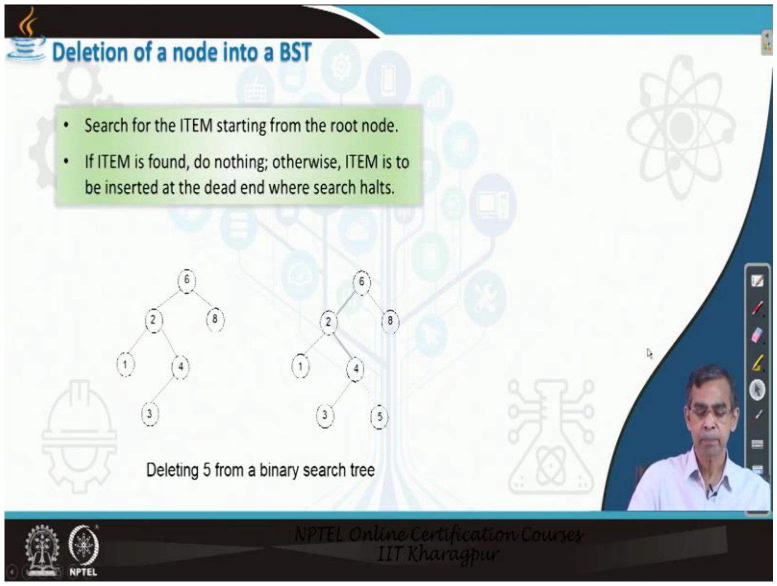
So this basically take that, whether we will follow this path. If it is not, then whether we have to follow this path. So it is the left, right traversal actually it is there. So this is basically left path like. If we does not follow, then we have to go to this path like this one. So this is operation that we can follow. And this basically the insertion operation as a whole, you can add this method into the class and then insertion operation is ready.

(Refer Slide Time: 9:50)



Method for Deletion

NPTEL Online Certification Courses
IIT Kharagpur

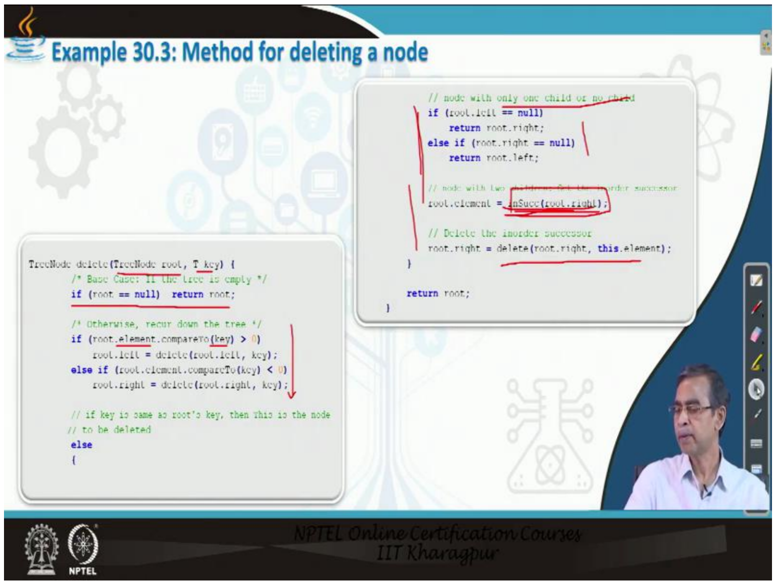


Deletion of a node into a BST

- Search for the ITEM starting from the root node.
- If ITEM is found, do nothing; otherwise, ITEM is to be inserted at the dead end where search halts.

Deleting 5 from a binary search tree

NPTEL Online Certification Courses
IIT Kharagpur



Example 30.3: Method for deleting a node

```java
TreeNode delete(TreeNode root, T key) {
        /* Base Case: If the tree is empty */
        if (root == null)  return root;

        /* Otherwise, recur down the tree */
        if (root.element.compareTo(key) > 0)
            root.left = delete(root.left, key);
        else if (root.element.compareTo(key) < 0)
            root.right = delete(root.right, key);

        // if key is same as root's key, then This is the node
       // to be deleted
        else
        {
            // node with only one child or no child
            if (root.left == null)
                return root.right;
            else if (root.right == null)
                return root.left;

            // node with two children: Get the inorder successor
            root.element = inSucc(root.right);

            // Delete the inorder successor
            root.right = delete(root.right, this.element);
        }

        return root;
    }
```

Our next task is basically the deletion operation. Deletion operation is very simple here. Deletion operation, again the three cases that as I told you. So this is one case like, and other cases are there. Now let us see how the different cases of deletion can be implemented. Here I have defined the different cases of there. Now here if you see, so the deletion operation method is basically the start node and then key. The key, that means that is the target value that needs to be removed from the T. Now here this is the case, situation is that if T is empty, then we do not have to do anything.

So this is the check that the T presently is to contains any elements or not. If it is not there, then this key should be compared with the current element, starting from the root actually, right? And then if it is greater than 0, then we just go to the left. If it is less than 0, then we go to the right. So it is basically go to the point where the key is present. So this basically is a recursive procedure, because it is starting from the root, go to the next, and is a recursive method. So recursively it will go.

Now if you do not find, then we have to come here, that is okay. This is the case 2 we can say. Case 2 or case 1, it is applicable. So that node has either one child or no child it is there. Now, so this case basically comes to this one. If it does not contain any child or one child, then we have to perform this operation. That means hitting the link of each parents like. So it basically does this one. Now on the other hand, this is the second, third case. That means case 3. That mean it have both the children.

Now if it has both the children, then what we have to do is that, we have to find the minimum value next to that node. So this method, actually there is another method is required. This method is called the minimum value of a node is basically called inorder successor. That mean if you traverse the tree in an inorder successor, for a given node, the next node is basically the inorder successor in that node because 24, next higher value of the 24, maybe say 26, then next value maybe 29, so in inorder traversal if we see, 26 will come after 24.

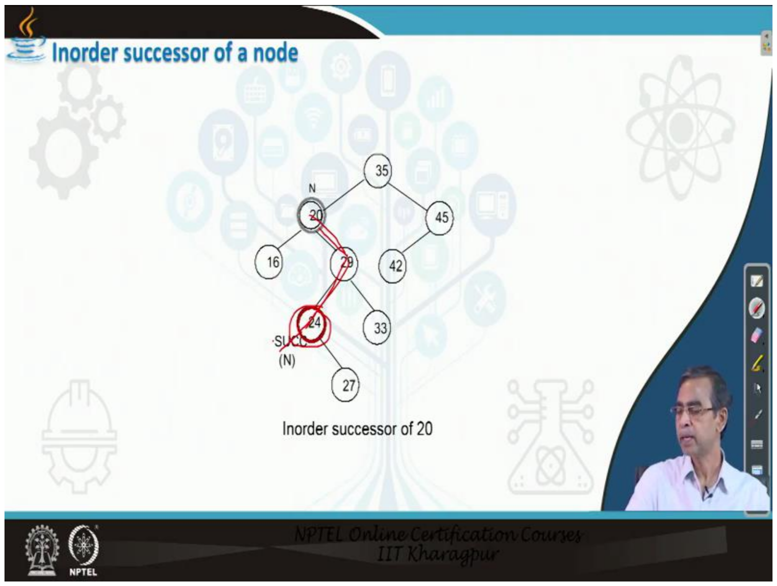
So this sense it is called. So finding mean value of the node to be deleted is basically finding the inorder successor. That means as if you are performing the inorder traversals and finding the next node of the current node it is there. Now so if we find this inorder successor, where it is there, we have to go there, and then you have to delete that inorder successor elements from there. So it is the procedure, and this program will work for you taking care of all the three cases, and you can check that this program is working.

And you, you are advised to run the program, and check it is, that it is working. If you find any mistakes and everything, then you have to do the correction. But I feel that this program are tested, and you will find it, ready, I mean easily executable.

(Return Slide Time: 13:03)

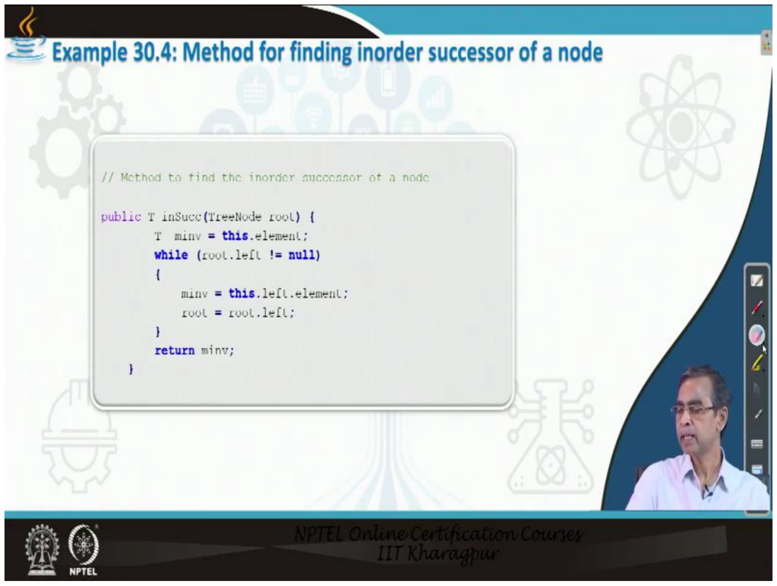
Method for Inorder Successor

NPTEL Online Certification Courses
IIT Kharagpur


Inorder successor of a node

Inorder successor of 20

NPTEL Online Certification Courses
IIT Kharagpur



Example 30.4: Method for finding inorder successor of a node

```
// Method to find the inorder successor of a node

public T inSucc(TreeNode root) {
        T  minv = this.element;
        while (root.left != null)
        {
            minv = this.left.element;
            root = root.left;
        }
        return minv;
    }
```

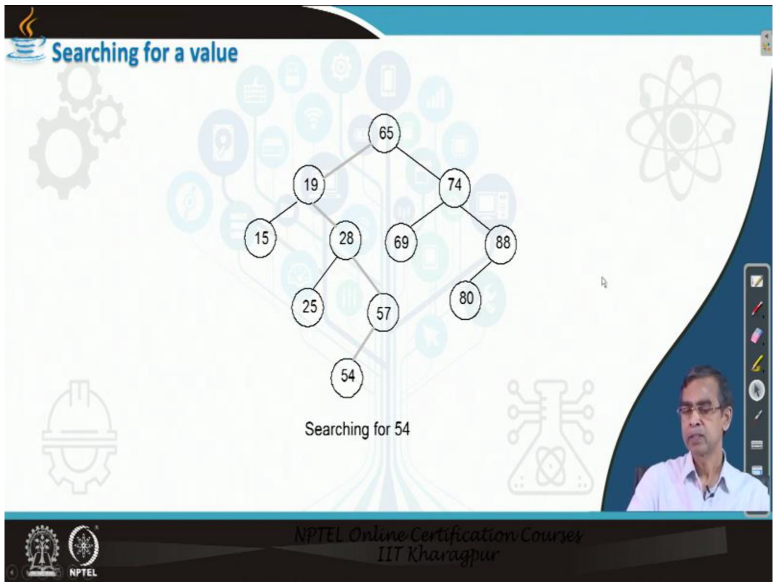
Now, so here is a method for inorder successor, how you can find it. For example, 24 is the inorder success of 20. Idea you see like this, the programming is very simple actually. So go to right, and then go left left left until you get the, gets it left sub tree is empty. So it is basically right, go right and then go left left left. We stop here, because it is left sub tree is null. So this is basically the inorder successor of 20 actually. So procedure is like this, right?

So that procedure can be easily implemented and this is the program that we have, we have written and then you can check this program. This program is basically the same procedure as I told you, go to the right and then left left left, until you can find it and then finally finding the inorder successor. Here main tree is basically inorder successor of the nodes that needs to be considered. So this is the procedure that you can consider for the deletion operation, deletion operation.

(Refer Slide Time: 14:12)

Method for Searching

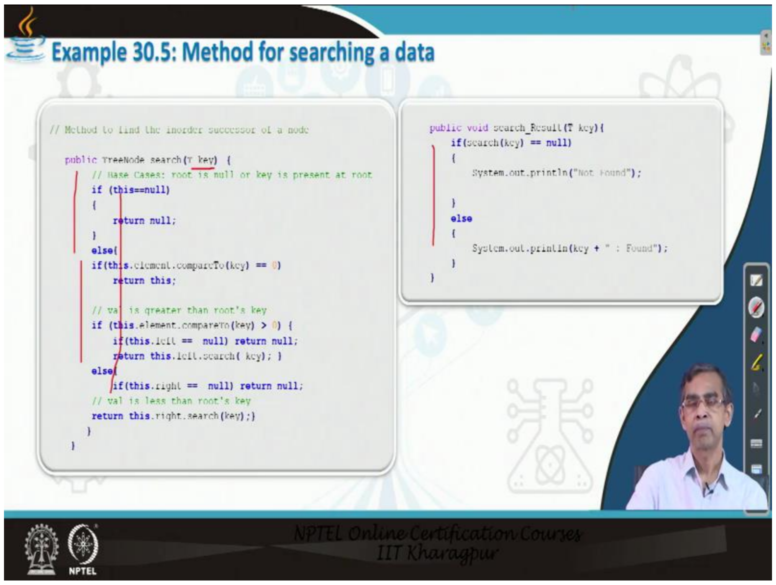
NPTEL Online Certification Courses
IIT Kharagpur



Searching for a value

Searching for 54

NPTEL Online Certification Courses
IIT Kharagpur



Example 30.5: Method for searching a data

```
// Method to find the inorder successor of a node

public TreeNode search(T key)  {
    // Base Cases: root is null or key is present at root
    if (this==null)
    {
        return null;
    }
    else{
    if(this.element.compareTo(key) == 0)
        return this;

    // val is greater than root's key
    if (this.element.compareTo(key) > 0) {
        if(this.left ==  null) return null;
        return this.left.search( key); }
    else{
        if(this.right ==  null) return null;
    // val is less than root's key
    return this.right.search(key);}
   }
 }
```

```
public void search_Result(T key){
    if(search(key) == null)
    {
        System.out.println("Not Found");

    }
    else
    {
        System.out.println(key + " : Found");
    }
}
```

And then searching operation is very simple again. The algorithm of searching is basically either you have to follow left or right, until we reach either end or you can find the elements itself. So that code is written here. So we can start that, okay key needs to be search, we can check this, always here whether T is empty or not. If T is empty, we do not have to do anything.

Otherwise we can search from either left direction or right direction depending on the value of the node to be searched. And accordingly you can find it. So this method is again simple algorithm that we have once mentioned for the searching. It is implemented here within programming. So my suggestion is that first consult the algorithm, following the algorithm and then if you follow the code, you will find little bit easy to understand the code.

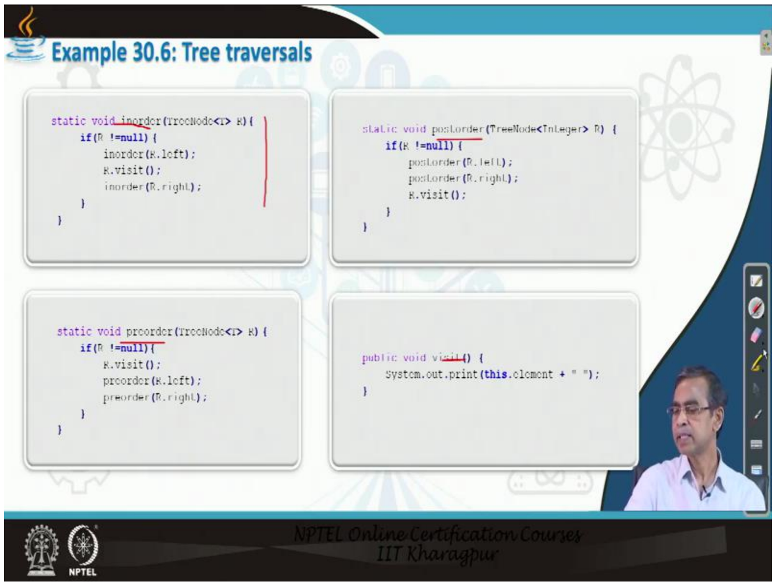
For starting with code it is very difficult. So my suggestion is that, first check the algorithm and then come to the code, and then try to run the code. And then you will be able to understand how programming takes place. But you know programming is done by once and you have to allow it, but best idea would be that you can start writing the program of your own, starting with the algorithm itself. So forgo all this code, whatever it is given here. But you can get an idea about that. You can just okay, you should be happy with your own code.

And it has advantages in many aspect that you can, it will develop the coding skill in you. And at the same time, it will give you better understanding about this logic and algorithm implementation. So this is the idea about the different operations that I have mentioned. Now as an extension you can think about the other operation like merging, or traversals also you can implement.

(Refer Slide Time: 15:50)

Method for Traversal

NPTEL Online Certification Courses
IIT Kharagpur



Example 30.6: Tree traversals

```
static void inorder(TreeNode<T> R){
    if(R !=null) {
        inorder(R.left);
        R.visit();
        inorder(R.right);
    }
}
```

```
static void postorder(TreeNode<Integer> R) {
    if(R !=null) {
        postorder(R.left);
        postorder(R.right);
        R.visit();
    }
}
```

```
static void preorder(TreeNode<T> R) {
    if(R !=null){
        R.visit();
        preorder(R.left);
        preorder(R.right);
    }
}
```

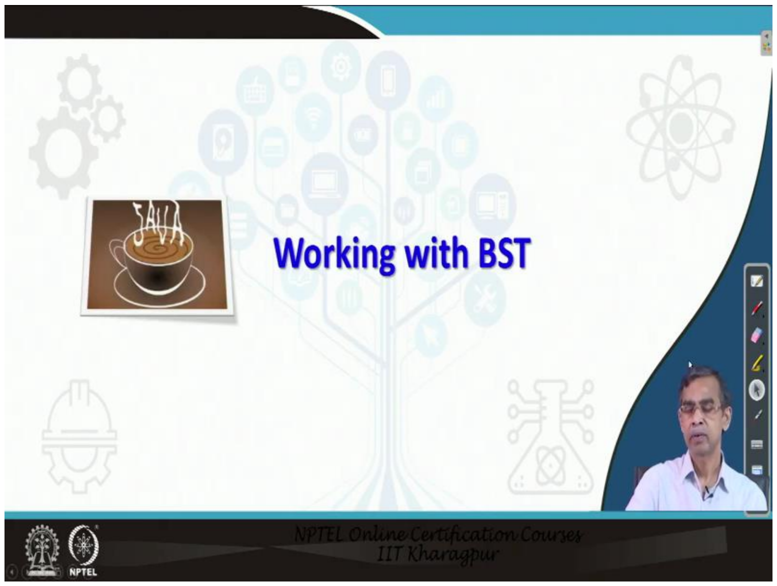
```
public void visit() {
    System.out.print(this.element + " ");
}
```

NPTEL Online Certification Courses
IIT Kharagpur

I can give you a quick idea about how the different traversals likely inorder, postorder, preorder can be implemented. It is very simple easy code. Again you can write the code yourself. So this is for the matter of practice actually. So here this is the code, as you see this is a recursive method we have written here for the inorder traversals. And then preorder traversals, and postorder traversals. And visit basically if you want to print the entire tree, then you can see the visit is there.

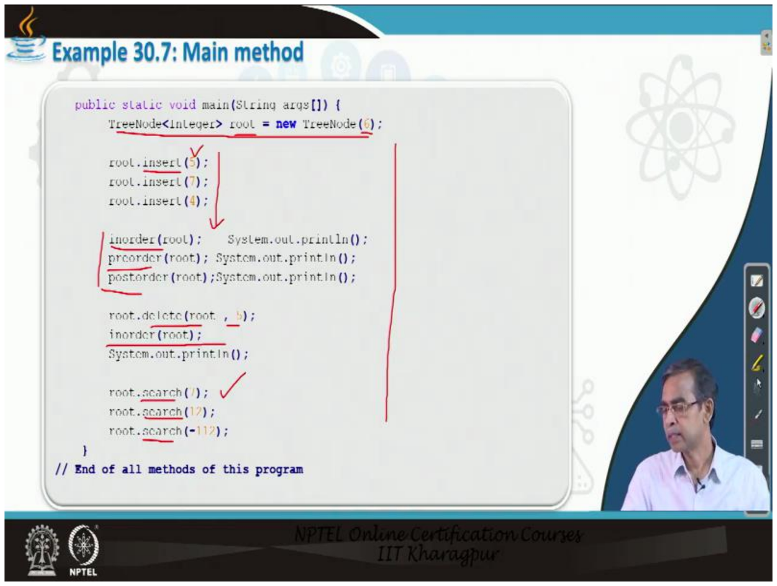
Anyway, so basically visit means printing the elements, at the current node, whatever is there. So these are the complete code that you can include and in the class declaration there and then you can run it. Now let us see the master program who is basically whatever the code that we have developed to test, how they are running with the different, what is called the context or different or activities like.

(Refer Slide Time: 16:47)



Working with BST

NPTEL Online Certification Courses
IIT Kharagpur



Example 30.7: Main method

```
public static void main(String args[]) {
    TreeNode<Integer> root = new TreeNode(6);

    root.insert(5);
    root.insert(7);
    root.insert(4);

    inorder(root);    System.out.println();
    preorder(root); System.out.println();
    postorder(root);System.out.println();

    root.delete(root , 5);
    inorder(root);
    System.out.println();

    root.search(7);
    root.search(12);
    root.search(-112);
  }
// End of all methods of this program
```

NPTEL Online Certification Courses
IIT Kharagpur

So here is an example. This example is the main method I have declared here. This is belong to the, maybe this method is the next part of the class declaration that we have here. So let us have the main method at the end of the class declaration, so that you can run this method very easily. Now, here is the main method. So we create a binary tree, the name of the binary tree, rather we can say that the starting point of the binary tree is root here. And we create, we 6 is the initial node given to it, so this is the initial node.

So that means this will create a single node in the binary search tree, and the root node is 6. So in this case, root node is 6. Now, we just called the insert method and that means 5 we want to insert. So we insert this 5 means it will insert at the left part of the node, then it will be inserted. Next 7, 4. In order you can carry out with more insertion with some other elements. Again the same elements which you already inserted beforehand also, and you can check that your program is working and how it is working like.

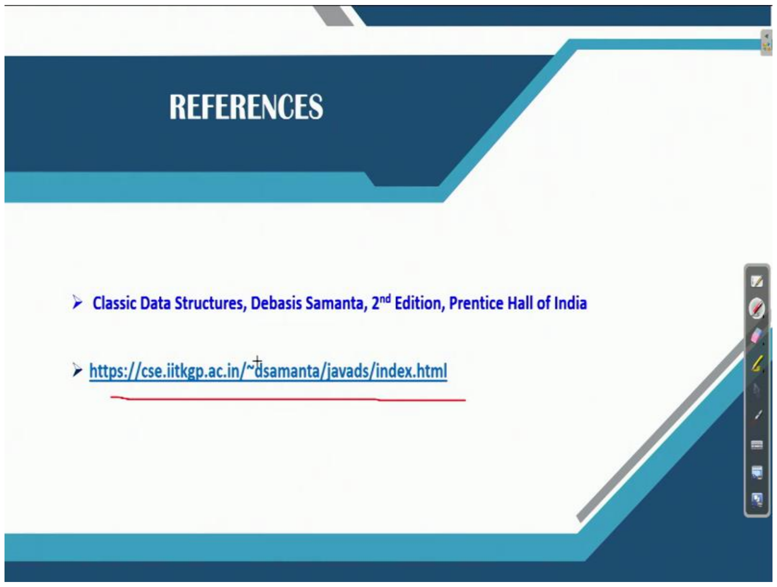
And after the insertion, then here I have given an idea about traversing. So three different traversals inorder, preorder, postorder, it will basically give the value, that it basically order that it visit all the nodes in the Binary Search Tree. So we have learnt about how a tree can be created, initially with starting node and then insertion can be taken place one by one and then the traversals.

Now we can insert few more nodes or delete, and insert delete, whatever the order OA, you can randomly, you can choose it. Now here the delete node from the root node, the 5 to be deleted. And then after deletion again if you perfectly know traversal, you will be able to understand that 5 has been deleted successfully. And like this, so there are few more operation like search, we have already implemented. We can find with the different value, whether the element present there or not and so on, so on.

So this program can be experimented with many other situation of many different operations, insertion, deletion and searching, and then traversals, whatever altogether. Then you will be able to see that your tree data structures, rather Binary Search Tree data structures is ready to solve any application. So if you want to apply these program to solve many application, you just simply include or import these class into your program, where you want to apply it.

So this is the idea about the programming, and then idea about using this data structures, in fact in greater sense actually. So we have learnt about the different operations, the Binary search Tree, and the different cases that we have considered. And more practice is required, so that you can understand these things in a more better way.

(Refer Slide time: 19:58)



REFERENCES

- Classic Data Structures, Debasis Samanta, 2nd Edition, Prentice Hall of India
- https://cse.iitkgp.ac.in/~dsamanta/javads/index.html

And for further study, you can follow this book, where it can give you many other details which is not possible to cover or which is not covered in this discussion, rather more details inside of the tree structure mainly and then particularly Binary Search Tree can be can be found here. And this is the link I am maintaining in this link you can find all the programs, as well as the detailed discussion about actually the known version of this core, of this presentation also you can get it.

As well as the presentation slide also you will get it from there. So this way I made all the things available to you, so that you can learn it more better way. But learning again depends on programming, and then programming is very important. So I allow, I rather insist you to go for rigorous programming and then you can understand this concept much better. So with these things I want to stop it here for today. So Binary Search Tree is over.

Our next part of the discussion will include the another different type of tree like heap tree. Heap tree has the many important application, like Binary Search Tree. Binary Search Tree is a very important on data structures actually in the theory of computer science. Heap tree is also no inferior than the Binary Search Tree, it has also many application. So in the next lecture video, we will study Heap Tree. Thank you very much.