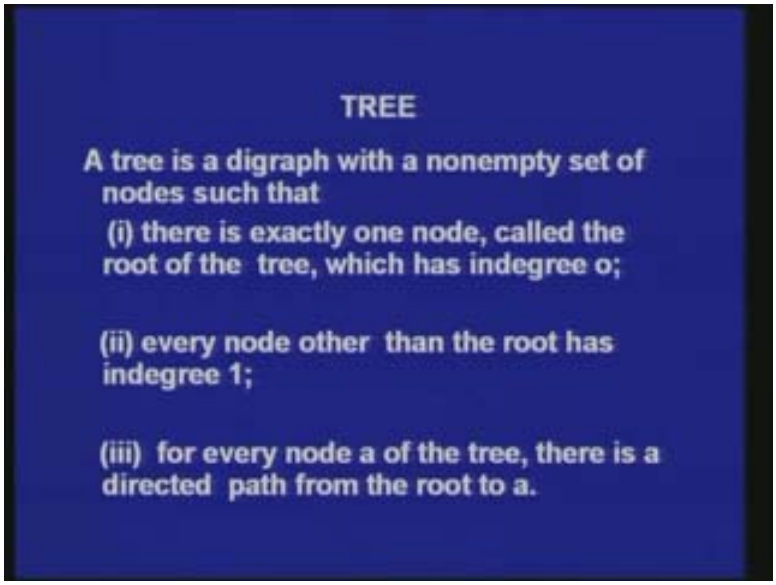
Discrete Mathematical Structures
Dr. Kamala Krithivasan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture # 16
Module -2
Trees

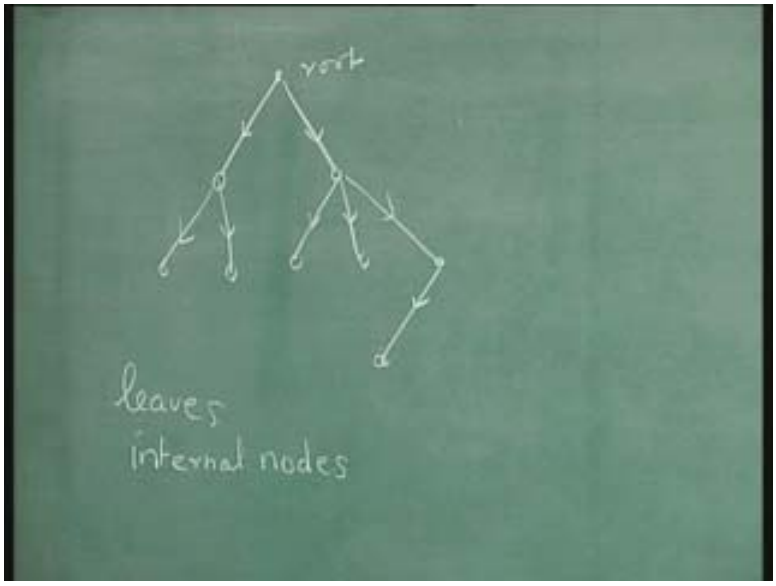
So today we shall consider about trees and search trees. A tree is a digraph with a nonempty set of nodes such that there is exactly one node called the root of the tree which has degree 0. Every node other than the root has indegree 1. For every node a of the tree there is a directed path from the root to a.

(Refer Slide Time: 1.10)



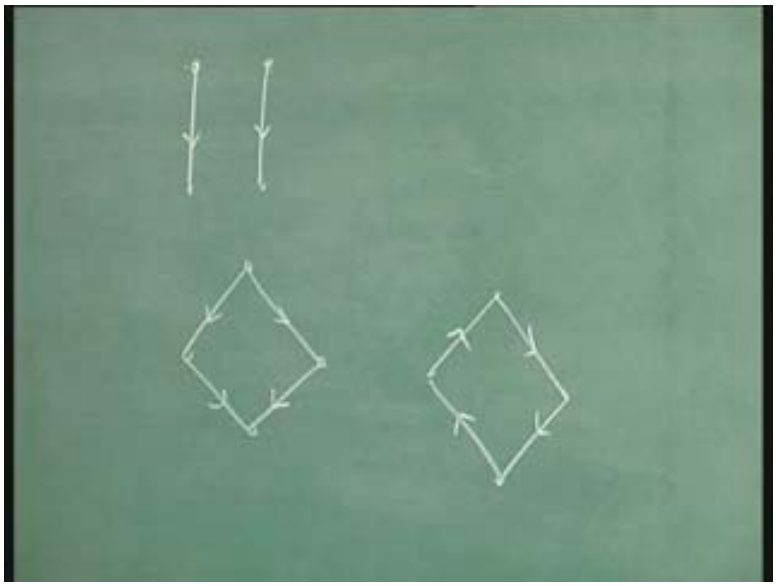
So a tree has a structure something like this; there is a root which as got indegree 0 and you may have other nodes like this. This is a tree, these are called leaves. The other nodes are called internal nodes. For every node there is a directed path from the root to the tree.

(Refer Slide Time: 2.12)



Now following are not trees, consider this. If you have something like this, this is not a tree because you are having two nodes with indegree 0 so that is not a tree. And look at this, this is also not a tree because you have a root no doubt but this node has indegree 2 and so this is not a tree. And similarly, if you take something like this directed in this manner this is not a tree because there is no node with indegree 0.

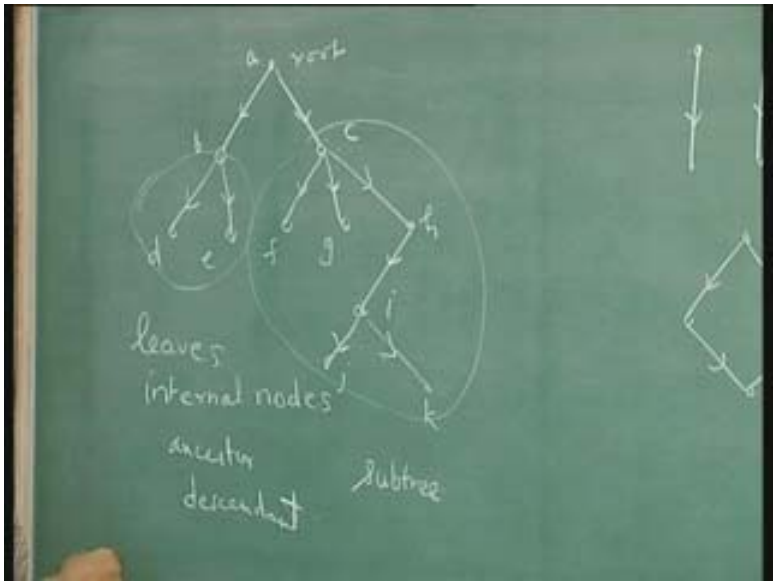
(Refer Slide Time: 3.06)



There is no root at all in this so this is not a tree. So a tree has a structure something like this. And you may have more and more levels you may have something. Now for this node these are the sons so if I label them as a, b, c, d, e, f, g, h, i, j, k then for c f, g, h are

the sons and for i h is the father, for d and e b is the father and so on. a has two sons b and c, c has three sons and so on. Now a is an ancestor of i. In the path it comes before i from the root, c is also an ancestor, i is called a descendant. And look at this portion this is called a subtree and the subtree has root at c. Similarly, if you take this, this is a subtree and this subtree has a root at b. So these are some of the terminologies associated with tree.

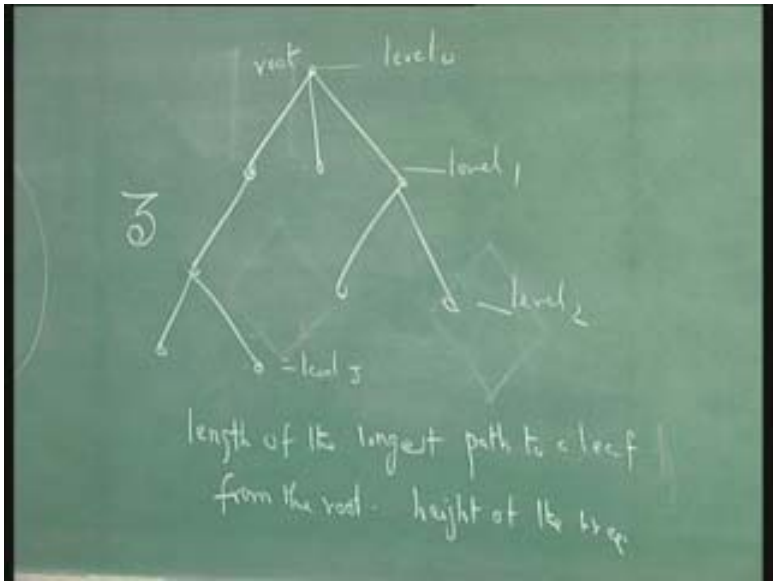
(Refer Slide Time: 4.36)



And generally when you draw a tree you do not draw the arcs you just draw a tree like this, just draw like this. It is supposed to be directed in this way and this is the root. Now what is the height of this tree?

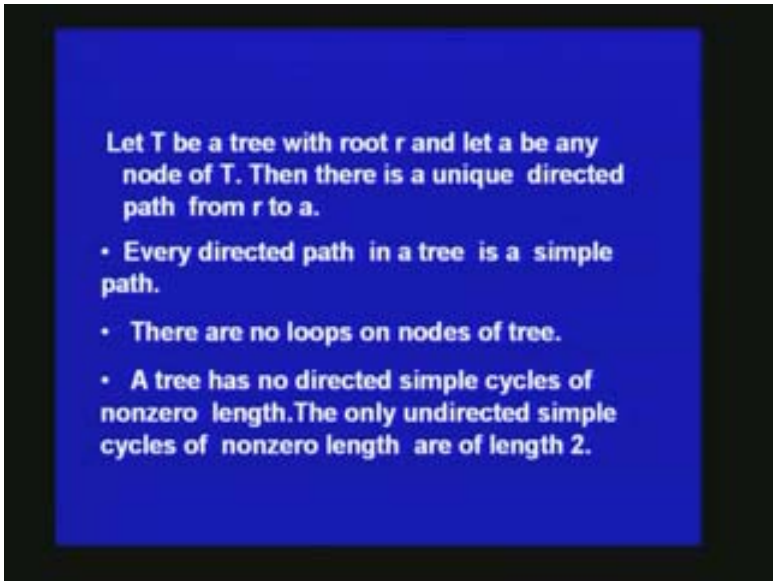
The height of the tree is the length of the longest path to a leaf from the root. This is called the height of the tree. So look at this, the height of this tree is 3. Now the root is suppose to be at level 0 and these three nodes are at level 1, these three nodes are at level 2, these two are at level 3 the height is 3.

(Refer Slide Time: 6.15)



So these are some terminologies associated with the tree. Now, let us consider this, let T be a tree with root r and let a be any node of T. Then there is a unique directed path from r to a.

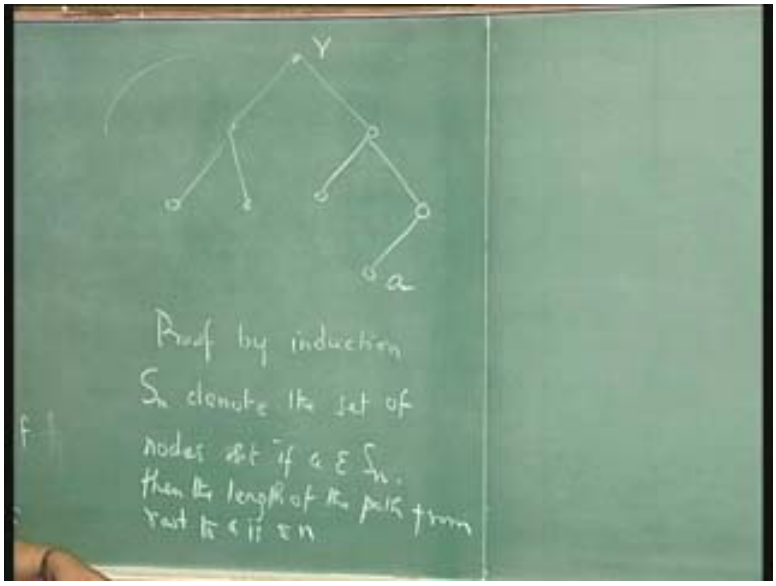
(Refer Slide Time: 6.16)



You have a tree T something like this, then by definition there is a path a is any node there is a path to a from the root. But we want to show that there is a unique path.
How do we prove that?
That there is a directed path from root to a is clear, by definition it is there and it is also a simple path. We can very easily see that it is a simple path. But that path is unique, how

do we prove that? You can proof by induction, proof by induction. You have a tree say some tree like this it is only a small tree I have drawn but generally any tree and let S_n denote the set of nodes such that if a belongs to S_n then the length of the path from root to a is less than or equal to n.

(Refer Slide Time: 8.30)

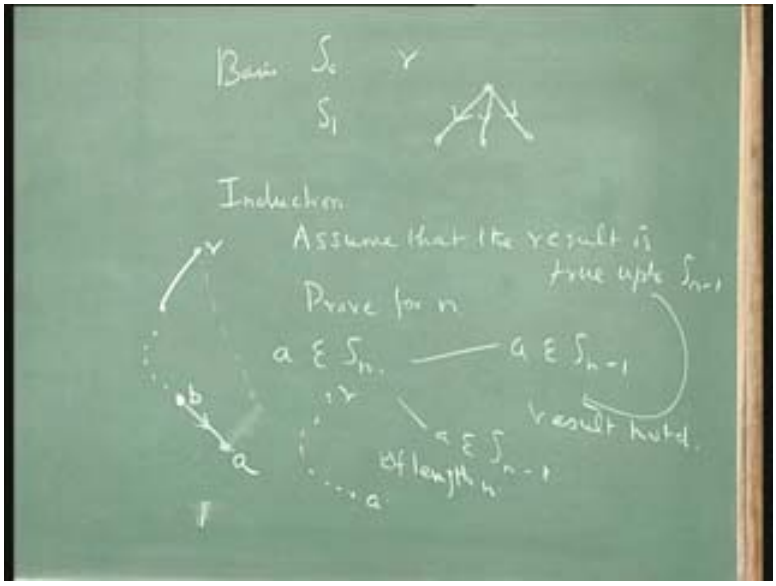


Now the basis clause you can prove like, is S_0 means it is only the root, for the root there is only one path of length 0 from itself to itself so that is unique. If you take S_1 it denotes nodes which are sons of the root and you can very easily see that there will be only one directed path the path is unique to them so that is okay. Now induction portion, assume that the result is true up to S_n minus 1 that is up to S_n minus 1.That is, if the node is such that the length of the path from the root to that node is less than OR is equal to n minus 1 then there is a unique directed path from the root to that node. Then proof for n, you have to prove for n.

Suppose a belongs to S_n there are two possibilities; one is a belongs to S_n minus 1 that is the length of the path from the root to a is less than OR is equal to n minus 1, then by the induction hypothesis the result holds. The other thing is, a does not belong to S_n minus 1 that means the path from the root to a is of length m, from root the path to a is of length n. That means there is a path like this. Root to some node and then to this node, this is a, some node b this path is of length n minus 1 and from b to a you have a path.

Suppose this path is not unique there can be another path say if there is another path then the indegree of this will be 2 but you can have only indegree 1 for this node a. So indegree 1 means there is only one arc like this entering into a. And this node b will belong to S_n minus 1 that is the length of the path from the root to this node is of length of n minus 1 and this path is unique. So this path is unique so like that you can prove. And by induction hypothesis the result is proved. Now arithmetic expressions you can express as trees.

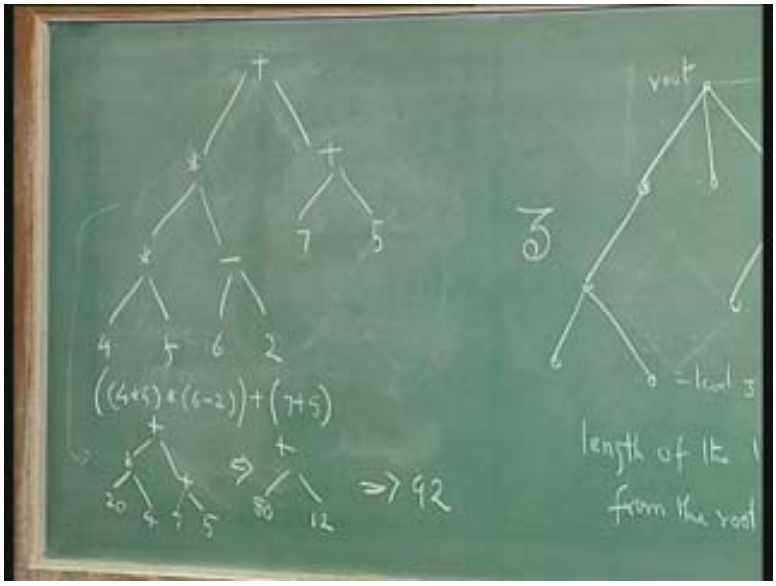
(Refer Slide Time: 12.02)



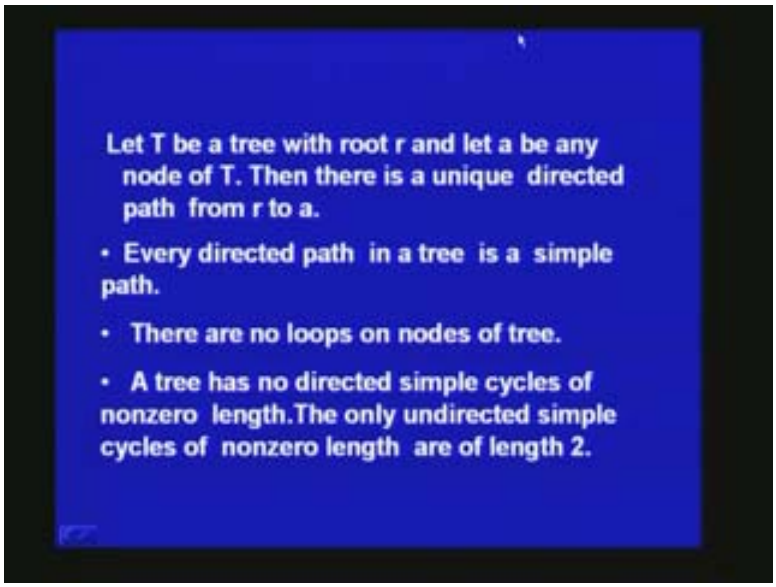
Let us consider some arithmetic expressions and see how they can be represented as trees. Look at this tree where the internal nodes represent the operators plus star star some 4 5 minus 6 2 plus 7 5.

What does this tree represent?
This tree represents an arithmetic expression and you can write it like this; 4 star 5 star 6 minus 2 that is this portion and it should be added to 7 plus 5. This arithmetic expression is represented as a tree like this. Now, how do you evaluate this? You can evaluate them level by level. So if you try to evaluate this 4 star 5 is 20. So this will be plus star plus and here the evaluation gives you 20. And here 6 minus 2 gives you 4 and plus 7 5 you have. So this evaluates to 20 cross 4 is 80 and 7 plus 5 is 12, now this evaluates to 80 plus 12 which is 92 so the arithmetic expression evaluates to 92.

(Refer Slide Time: 14.22)



(Refer Slide Time: 15.58)



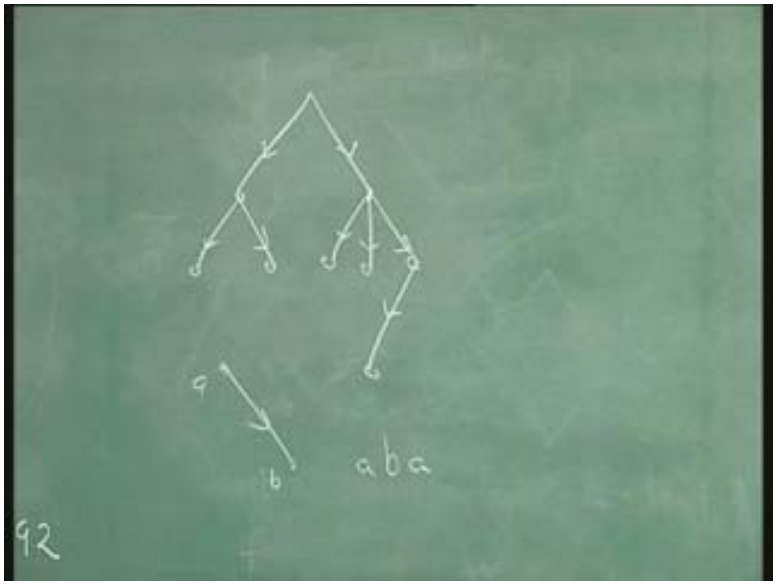
A compiler is a large program which translates a source language high level language in to a machine port. And there are two parts of the compiler one is the analysis part and another is the synthesis part. In the analysis part the source program is analyzed and kept inside the computer in an internal form. The internal form could be a triple or a quadruple or a tree or a four fixed notation or a three fixed notation something like that.

Let us see what is four fixed notation and three fixed notation a little bit later. Now when it is represented inside the computer like this in the internal form then the code is generated. But essentially the evaluation is done in this manner and for that the proper

code is generated. You will learn more about the compiler design theory course. And the remaining part notice that the arithmetic expression can be represented as a tree like this. And the internal nodes or operators leaves or numbers or variable whatever it is, it could be something like a b and then a can be given a particular value b can be given a particular value and so on. Now you will see these points very clearly.

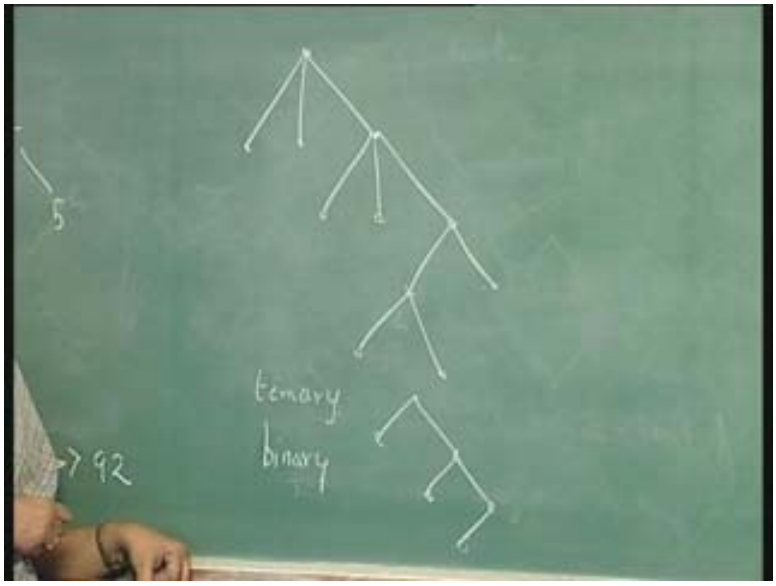
Every directed path in a tree is a simple path because no node will be repeated. There are no loops on the nodes there are no self loops in the nodes and the tree has no directed simple cycles of non zero length. So, if you have a tree like this usually we do not write the direction of the arcs explicitly but without them we write but suppose you also draw the arcs. Look at this, every path is a simple path and there are no self loops and there is no directed cycle and the other thing is the only undirected simple cycles of non zero length or of length two that is between these two. Suppose you have some arc like this ab you can look at it as a undirected cycle aba that is what the set by this. This is not a major point but the point is there are no cycles here there are no self loops and every path is a simple path.

(Refer Slide Time: 17.26)



Now, when you look at a tree like this; this has two sons, this also has two sons, this has three sons, this has three sons, this is a ternary tree. In a ternary tree each node will have three sons, two sons, one son or zero. No node will have more than three sons. When you look at a binary tree, this is a binary tree, each node will have two sons, one son or zero sons, the leaves have zero sons, this node has got one son, this has got two sons, this has got two sons and so on. No node will have more than two sons that is called a binary tree.

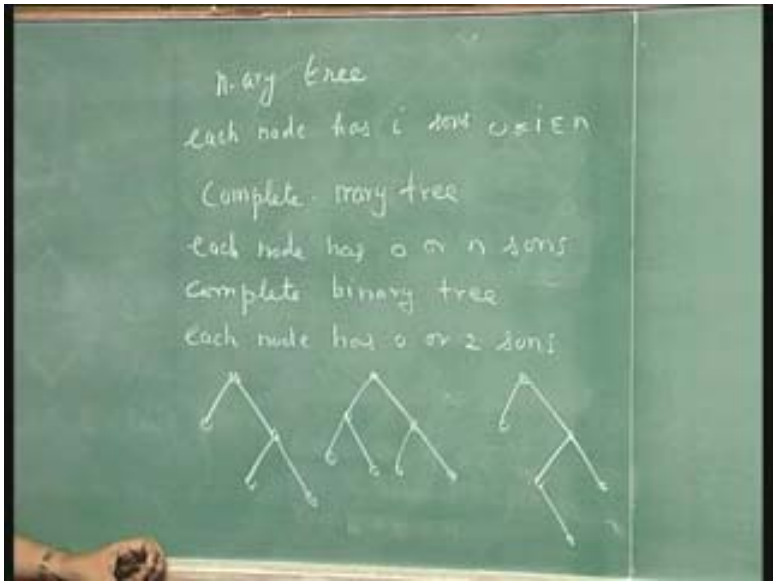
(Refer Slide Time: 18.35)



In general, n-ary tree means each node will have n sons or less. Now, what is a complete n-ary tree?
N-ary tree means each node has i sons, i can vary from 0 to n. A complete n-ary tree means each node has 0 or n sons. So, if you take the root it can have n sons and if you take any other node it can have n sons otherwise it will be a leaf. This is called a complete n-ary tree. What is a complete binary tree? In a complete binary tree each node has zero or two sons.

For example look at this; this is a complete binary tree. Look at this tree; this is also a complete binary tree. Look at this; this is not a complete binary tree because this node has only one son so this is not a complete binary tree. It is a binary tree but not a complete binary tree.

(Refer Slide Time: 20.50)



So binary trees are very much used for search purposes, ternary trees are also used. Let us see how we can use binary trees as search trees. Before that what can you say about the height of a binary tree? So some results about the height of the binary tree: If T is a binary tree of height h and with n nodes then n is between h plus 1 and 2 power h plus 1 minus 1. Moreover, there exist binary trees in which these bounds are attained.

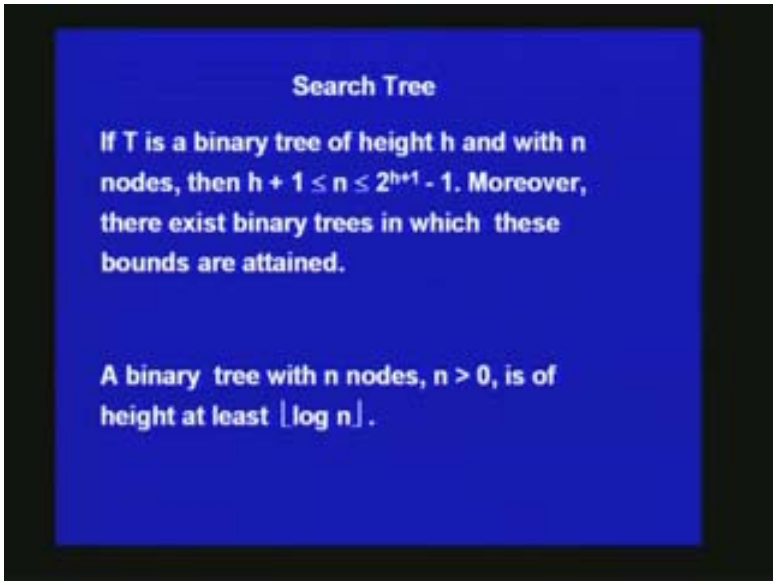
How do you prove that?
You have a binary tree, let me draw a binary tree, this is a binary tree.
How many nodes are there?
There are 1, 2, 3, 4, 5, 6, 7, 8 so 8 nodes are there.
What is the height of this tree?
It is 3, n is 8 and h is 3 here in this example.

Now, in general n is between h plus 1 and 2 power h plus 1 minus 1. Let us consider the worst case scenario, you can have a binary tree like this; a binary tree if it has two sons like this a node has two sons this is called the left son and this is called the right son. So you can have a tree like this where there are no right sons at all. Then what is the number of nodes here? n is 4 here and h is 3. In the worst case n can be h plus 1 this is the one. The other way round, every node has got two sons x so at every level so you may have a tree like this, a binary tree. This is a complete binary tree with height 3, this is also complete, please remember that this is also a complete binary tree. To distinguish between complete binary trees like this and this sometimes this is called a full binary tree. But that is not a very common terminology but some books use that notation.

So what is the number of nodes here?
The number of nodes is 15 n is 15 and h is 3.
So what is 2 power h plus 1?

Here, 2 power h plus 1 is 16 and 2 power h plus 1 minus 1 is 15. So in this case you get the maximum number of nodes for this height. For height 3 the minimum number of nodes you can get is 4 and the maximum number of nodes you can get is 2 power h plus 1 minus 1 is 15 you can have. You can also get like this, if the height is at level 0 you have one node, at level 1 you have two, at level 2 you have two squared nodes, at level three you have two cubed and so on.

(Refer Slide Time: 24.51)

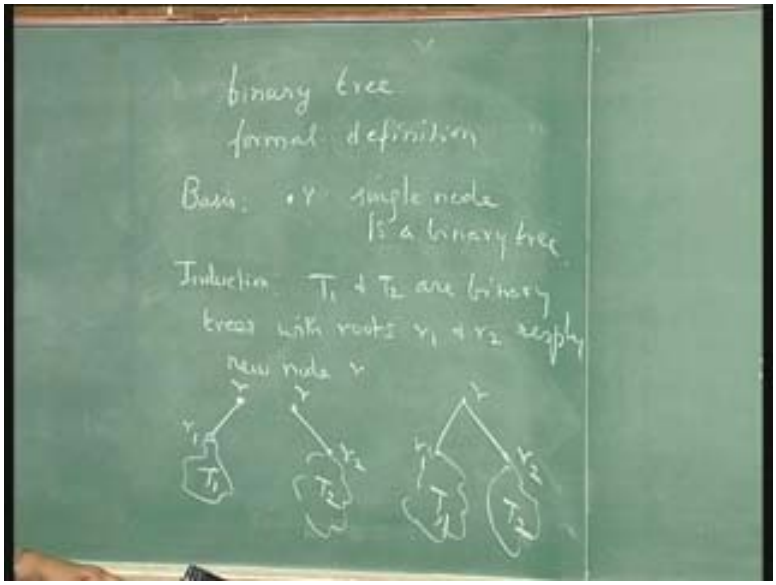


So in general if the height is h, I will write it here the height is h the maximum node you can have is like this, it is a geometric series sum and it is the sum of 2 to the power h plus 1 minus 1. So we have this result and the next one is the binary tree with n nodes the height is at least log n that you can very easily see. For 7 nodes you can have height 2 tree, for 8 nodes the minimum will be 3, for 9 nodes the minimum has to be 3, for 15 nodes you can have a tree with height h, for 16 nodes the minimum height required is 4..
If you have 16 nodes there should be something like this at one position the height will become 4. So you can very easily see that if there are n nodes the minimum height of the tree will be log n taking the floor function.

Now, all these things we have seen but how do you formally define a binary tree? How do you formally define a binary tree?
Formal definition: we have seen that sets can be defined inductively. So we can use an inductive definition to define a binary tree. So the basis part will be a root node alone like this or single node is a binary tree. Then the induction clause you write like this T_1 and T_2 are binary trees already constructed with roots r_1 and r_2 respectively. Then have a new node r, take a new node r, then r with an arc to r_1 along with T_1, this is a binary tree. Then r with a right arc to r_2 and the tree T_2, this is a binary tree. And r along with an arc to r_1 and an arc to r_2 with the tree T_1 and with the tree T_2, this is also a binary tree. So the basis clause is you take a single node that is a binary tree with no arcs.

And the induction clause is defined like this: if you have two binary trees T_1 and T_2 with roots r_1 and r_2 respectively take a new node r then r along with an edge like this and T_1 is a binary tree. Then r along with an edge like this to r_2 and T_2 is a binary tree, r with an edge to directed edge from r to r_1 and r to r_2 along with this T_1 and T_2 like this is a binary tree. Of course when you define a set inductively you have to also use what is known as the extremal clause. So here you have to mention the extremal clause. Earlier we have seen that the extremal clause is the same for all. That is all binary trees can be obtained in this manner and nothing else will be obtained. So this is the formal way of defining a binary tree.

(Refer Slide Time: 28.58)



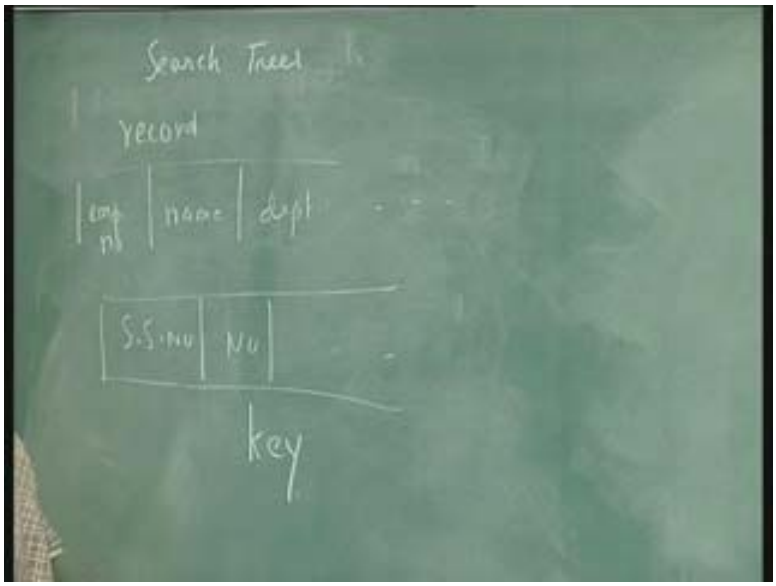
Now, these binary trees or ternary trees are very useful in data base and they are used as search trees. Data is stored as a record, a record will have several fields. For example, you have a record say it will have several fields, for example, employee in a company you want to store the records of all the employees, it will have the employee number name department whatever it is something like that.

If you take U.S. and so on you may also have social security number, name and so on. Now when you want search for some record it is better if the records are stored in the form of a tree so that the search takes less amount of time. If you have it as a linear record you have to search them one by one so it takes a lot of time. Here we have seen that when you have n nodes the height can be log n so in log n time you can search whereas if you store it as a linear record you will require search time n. Now when you want to search you have to search through a key, what is known as a key, the key should be unique for each record.

For example, if it is a department number or something, there may be so many employees in a department so that cannot serve as a key. Whereas if you take the social security

number of a person that is applicable in U.S. that is unique and in a company or even in IIT if you take each person has an employee number, employee identity. That is unique and no two persons will have the same employee number. So you can store according to that employee number which serves as the key. And you store the records and you have to search through the key. And you may use ternary trees or binary trees.

(Refer Slide Time: 31.11)



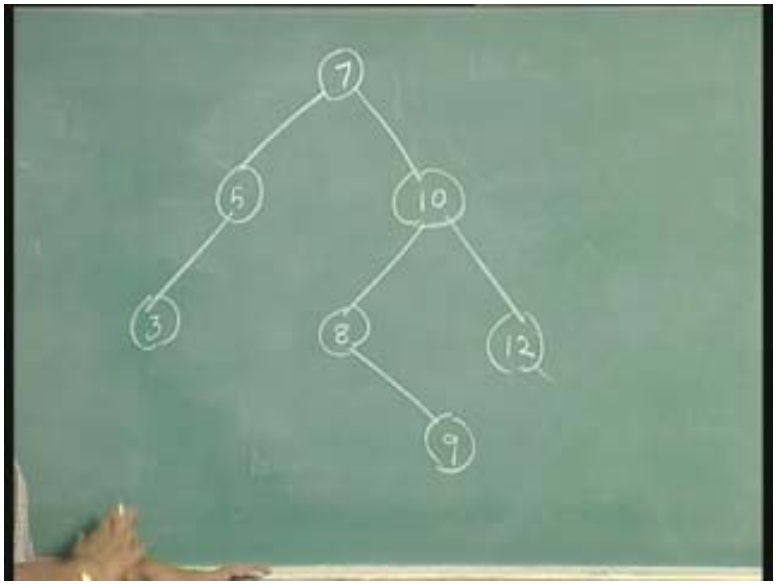
First let me see how we can store records using binary trees. For simplicity let us consider only some integers, let us take only integers. There are two ways of storing the records. First an example, you have this number say 7, 5, 3, 10, 8. Each record is stored in a node of the tree. This is the tree and each record is stored in a node, internal node or leaf. So if I want to search whether 9 is present how will I go about searching? First you go to the root, all elements less than 7 will be in the left subtree. All elements greater than 7 will be in the right subtree.

Suppose I want to search for 9 I go to the root I find that 7 is less than 9 so the next search I have to do in the right subtree so you go here. And here see 9, 9 is less than 10 so here I have to go to the left subtree and here I find 8, 9 is not there so that record is not found. Now you can add that 9 here 8 is stored here. So if you want add 9 you have to add like this; 9 is greater than 8 so it has to go to the right subtree of this. But then there will be some balancing also done. See the next one, suppose I want to add 14 I will add like this then 16 I have to add like this and the height of the tree will keep on increasing. So some balancing also will be done that will be explained in the data structures course. You will learn about that in the data structures course.

The main idea is the tree as a structure and how it is used. So you can store the record in each of the nodes and you can search like this. Each node will contain the key value. And if you are going to search for a value which is less than that you have to go to the left subtree. All the nodes on the left subtree will have values less than that. And all the nodes

on the right subtree will have values greater than that. This is the way all the data is stored. Another way of storing is, you store the records only at the leaves. And in the internal nodes you store discriminating values, they are called discriminators. The records are stored only at the root.

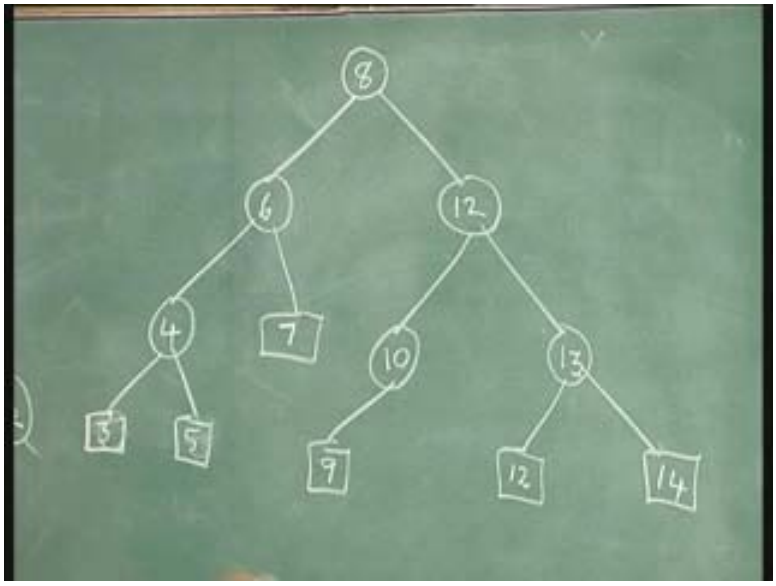
(Refer Slide Time: 34.38)



Let us see how this is done, take this. The leaf store the values the same value we can take 3, 5, 7 the values are stored like this. The root will contain a discriminator, it could be something like 8. Now, all values greater than or equal to 8 will be stored on the right subtree. They are stored at the leaves. All values greater than or equal to 8 are stored in the right subtree. Now in this node you must have a discriminator which tells you what are the things which go to this. Something like say you can have 12 here, all values greater than or equal to 12 will go to this subtree. And here you can have something like 13. So the value 12 is on this side, if the value is 13 or more it will be on the right subtree. And here you can have a value like say some 10 this is the discriminator. All values less than ten will be going to the left subtree greater than or equal to will go to the right subtree, there is no right subtree here. If you have a record say 10 or 11 it will go here.

Similarly, in this tree this value is 3 5 so you can have a discriminator 4 here. All values greater than or equal to 4 will go to the right subtree and all values less than 4 goes to the left subtree. And here you can have something like 6 so all values less than 6 go to this subtree and all values greater than or equal to 6 goes to the right subtree. But in this case the values are stored only at the leaves and the internal nodes contain the discriminators. This is one way of doing. So, there are two ways of storing the records and we have seen how to do that with binary trees.

(Refer Slide Time: 39.04)



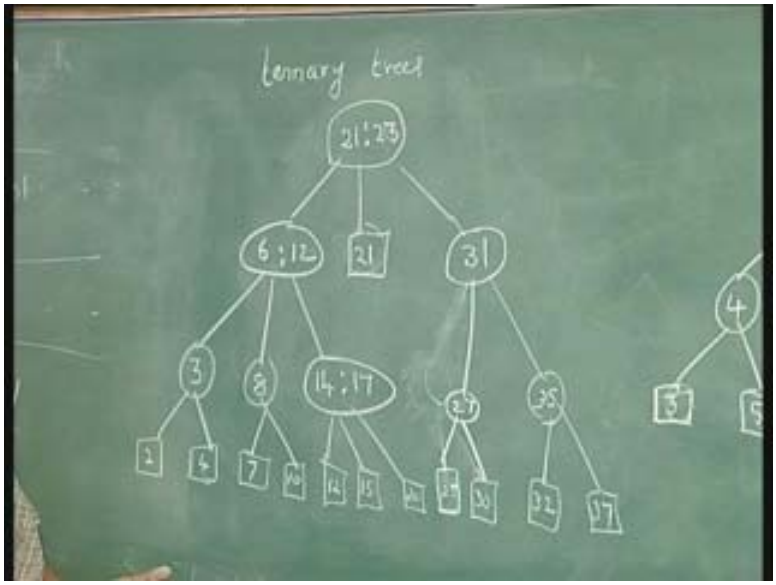
Let us consider ternary trees where the values are stored not at the internal nodes but only at the leaves. These ternary trees are also frequently used. Each node can have two or three sons. So, here we can have 3 it can have two sons or three sons. Now, again this has three sons and this has two sons, this you can take as a leaf and this may have leaves like this. This may have three sons like this, this as got one son and two sons. Some records you store here let us see how they are stored. It is 2, 4, 7, 10, 12, 15, 20, 21. And here it is 25, 30, 32, 37 and something like that.

Now, when you have three sons this node will have two discriminators. The first one here in this case can be something like 21. All values which are equal to 21 and the next value x say less than x will go to the middle subtree. Here from 25 we are having so I can have something like 23. So all value which is less than 21 will go to the left subtree and those which are equal to 21 and greater than 21 but less than 23 will go here, which are greater than or equal to 23 will go to the right subtree the third tree. And if a node has just two sons as in this case there will be one discriminator here. So the leaves are 31 here 32 and 37 you have. So one value can be 31 and all values less than 31 will go here and all values greater than or equal to 31 will go to this subtree.

Now here you are having 25 and 39, may be I should draw it in a slightly different way so that the left and the right are clearly seen. So here I should have a value such that it should be more than 25 and less than or equal to 30 something like 27 I can have so that the lesser value will go to the left subtree and the right greater than or equal to value goes to the right subtree. Here also there are only two sons so I can have a value say something like 35 here, so values less than 35 go here to the left subtree and values greater than or equal to 35 goes to the right subtree. Now this has got three sons so there should be two discriminating values here. And the middle subtree contains 7 and 10 so I can have something like 6 here. So all values greater than or equal to 6 but less than say twelve go to this subtree. And all values greater than or equal to 12 go to the right subtree.

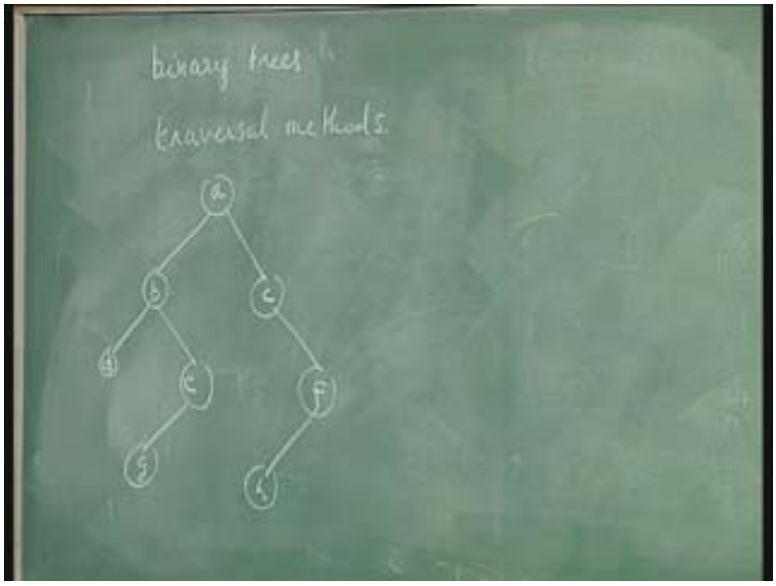
Now look at this, this again has got three sons, so here there should be two discriminating values. I can have something like 14 and 17. All values less than 14 will go to the first tree and all values greater than or equal to 14 but less than 17 will go to the middle subtree and greater than or equal to 17 will go to the right subtree. So the discriminating values we have to choose something like this. And here there are only two sons so I can choose a discriminating value like this. So the internal nodes only contain the discriminating values, if it has got three sons it will have two discriminating values and if it has got two sons it will have only one discriminating value. These ternary trees sometimes are called two three trees and this type of a tree is called a 2 3 tree. Each node can have two or three sons and this is not balanced but in some cases you expect all the leaves to be at the same level. In that case it is called a balanced tree. So, in general you would like to have balanced trees and all the data will be stored at the leaves and the internal nodes will have discriminating values. This is the main use of trees as a data structure.

(Refer Slide Time: 45.18)



Next let us consider binary trees and the way you can search through traversal, you can say traversal method. Suppose I have a tree like this where the nodes are denoted as a, b, c, d, e, f, g, h. There are three methods in which you can traverse the nodes. One is called the preorder traversal, another is called the inorder traversal and the third is called the postorder traversal.

(Refer Slide Time: 46.50)



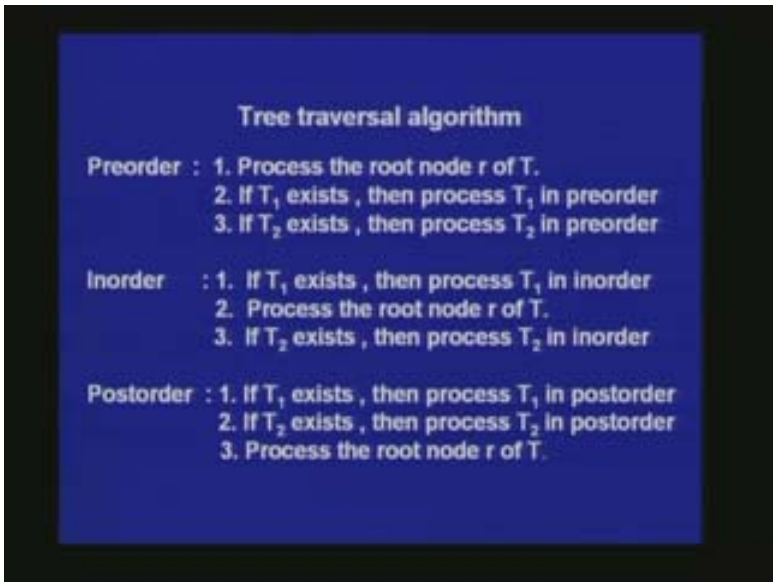
In the preorder traversal process the root r of the T then if the left subtree exists then process T_1 in preorder. And if T_2 exists then process T_2 in preorder. T_1 is the left subtree and T_2 is the right subtree. So, this is the tree you are having with node labels a, b, c, d, e, f, g, h. And if you want to traverse the tree in the preorder in what order you will search the root first then the left subtree and then the right subtree. So let us see how the nodes are traversed in the preorder manner in this tree. So node a will be visited first then these four nodes will be visited then these three nodes. And in these four nodes also you should follow the preorder method, the root first, the left subtree and the right subtree. So b will be visited first then the left subtree is just d then the right subtree e and g have to be visited.

And here again you have to follow the preorder traversal method, the root first, the left subtree and the right subtree, there is no right subtree here so e and g. Similarly, the right subtree also should be visited in the preorder manner the root there is no left subtree then the right subtree. So, for c, no left subtree and right subtree. And the right subtree again should be visited in the preorder manner root, first the left subtree and the right subtree, it is f and h. So the nodes will be visited in this order in the preorder traversal.

Let us see how the nodes will be visited in the inorder traversal. In the inorder traversal first the left subtree is visited and that is also visited in the inorder manner. If T_1 exists then process T_1 in inorder then process the root then the third one is if T_2 exists then process T_2 inorder. So, in the inorder traversal left subtree first then root then right subtree. So, if you want to traverse this particular tree in the inorder method in what order will the nodes will be visited?
First the left subtree then the root then the right subtree and the left subtree itself should be visited in the inorder manner, first the left subtree then the root and then right subtree. That is d first then b next then these two and these two again should be visited in the inorder manner, left subtree root and right subtree. So g and e will be visited in that order.

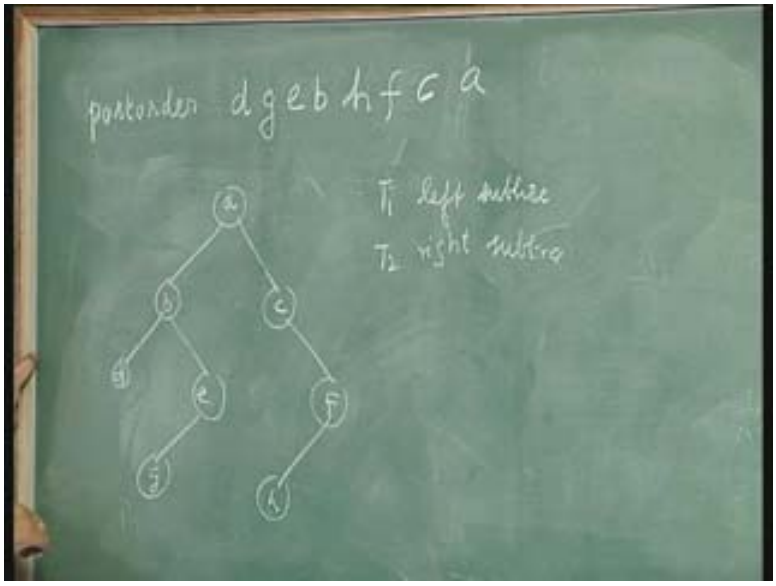
Now look at the right subtree, this also has to be visited in the inorder method that is left subtree, root, right subtree. There is no left subtree so c will be visited next then the right subtree. And this right subtree again has to be visited in the inorder manner so h first then f then if there is som thing in the right subtree there is no right subtree here so it will be h and f. So the nodes will be visited in this order in the inorder traversal method. There is one more method called the postorder method. in the postorder method the left subtree will be visited first then the right subtree then the root. So in the postorder method if T_1 exists then process T_1 in postorder then if T_2 exists then process T_2 in postorder, third process the root node r of the T.

(Refer Slide Time: 46.55)



So take this same example and if you want to traverse using postorder traversal in what order the nodes will be visited?
First it is the left subtree then the right subtree then the root. So a will come at the end. The left subtree itself if you take first the left subtree then the right subtree then the root so it is d then the right subtree again will be visited in the postorder method so it is g, and if there is something here then it is e, there is nothing here so it will be g and e, then b. So in this subtree the nodes will be visited in this order d, g, e, b and next the right subtree has to be visited that also has to be done in the postorder method. That is for this one there is no left subtree, right subtree has to be visited first then the root. So c will come here and this one again it has to be visited in the postorder method left subtree, right subtree, root. So there is no right subtree so h, f so the nodes will be visited in this order. So for binary trees you talk about these three methods of traversal; inorder, preorder and postorder.

(Refer Slide Time: 52.58)

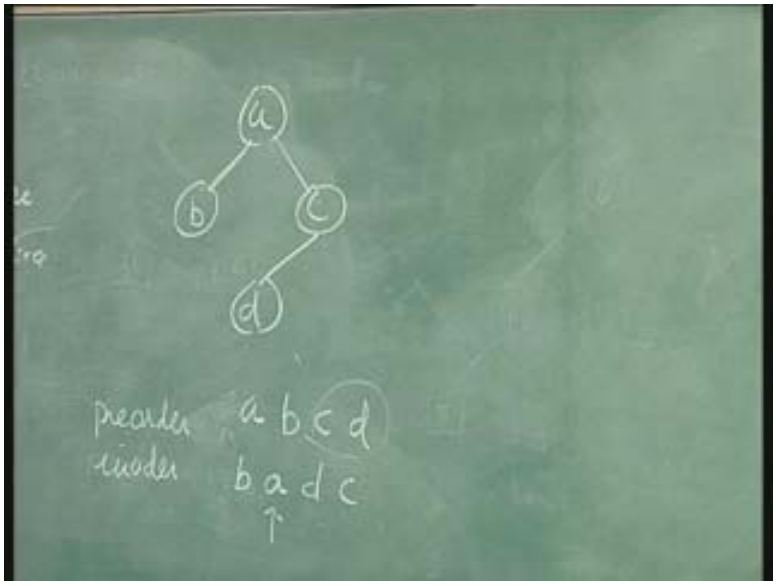


Now if you have the sequence of nodes visited in an inorder method and the preorder method from these sequences you can construct the tree. Similarly, if you have the sequences corresponding to inorder and postorder you can construct the tree in a unique manner. But if you have the sequence corresponding to preorder and postorder you cannot construct the tree in a unique manner.

For example, I have four nodes like this say a, b, c, d the preorder will be here will be a, b, c, d and the inorder will be b, a, d, c. From this you can construct the tree in a unique manner. How can you do that suppose I do not have this? I do not have this how can you construct the tree? The preorder a is the root. In the inorder a is the root so whatever is appearing on this side is left subtree and this on the right subtree. So b is on the left subtree there is only one node so it has to be like this.

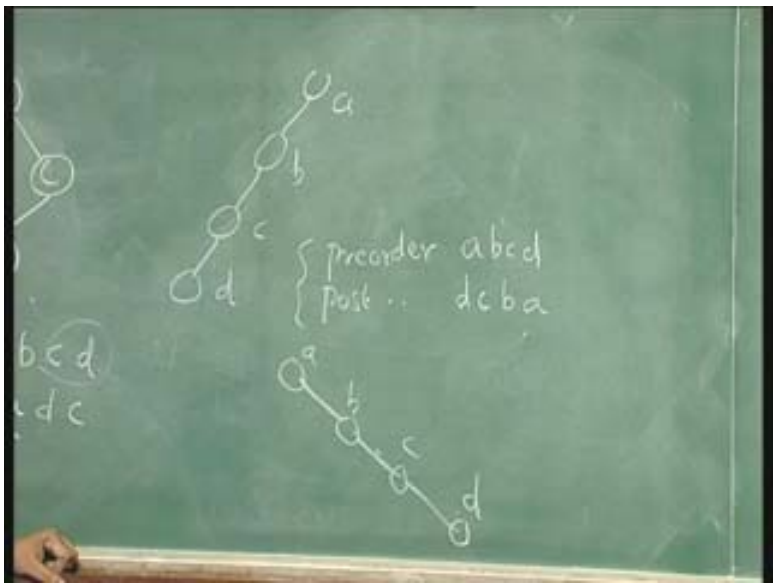
On the right subtree you have c and d. And c and d you have to be the right subtree, the first one will be the root of the right subtree so c will be here and where will be d? The d can be here or here right but this tells you that in the inorder left subtree first then root then right subtree that means d has to be left son here so you can draw like this. So from pre order and inorder you can the tree in a unique manner but you cannot do that if you have preorder and postorder.

(Refer Slide Time: 55.28)



For example, if you have something like this a, b, c, d what is the preorder?
The preorder is a, b, c, d, the postorder is d, c, b, a. If you are given this, the tree can be like this or it can be like this also a, b, c, d. What is the preorder for this? It is a, b, c, d and postorder is d, c, b, a. So the tree can be this or this. So from these two sequences you cannot construct the tree uniquely whereas if you are given inorder and preorder or inorder and postorder you can construct the tree in a unique manner.

(Refer Slide Time: 56.14)



And if you look at the arithmetic expressions the use of it will be very clear. Now we can see how these traversal methods are useful when you look at the trees representing

arithmetic expressions. We can also see some of the other concepts about trees like spanning trees, weighted spanning trees, etc in the next lecture.