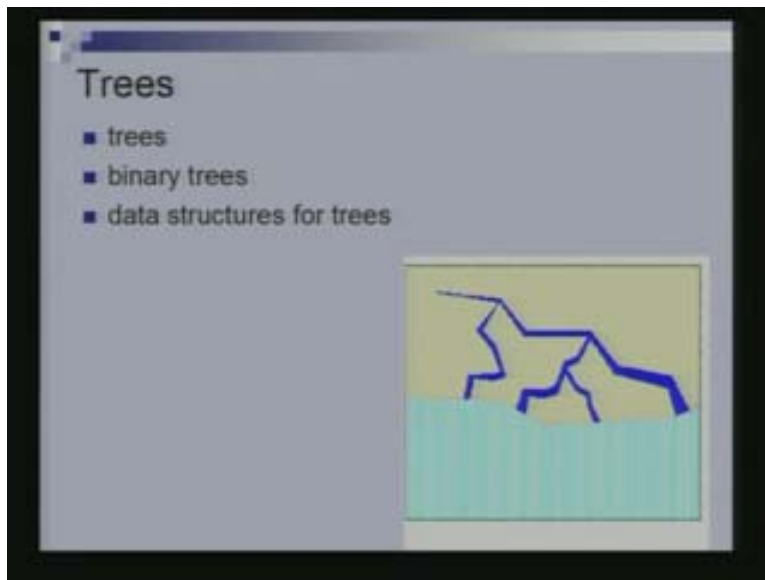


Data Structures and Algorithms
Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture – 6
Trees

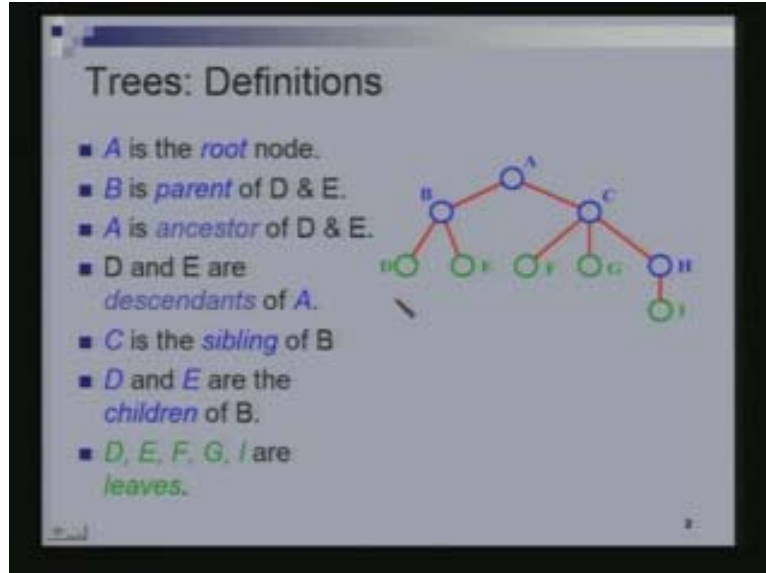
Last class we discussed about hashing. We saw few collision resolution techniques, chaining, double hashing linear programming and you also did a little bit of analysis of these collision resolution techniques. Today we are going to talk about trees. We are also going to look at binary trees and some data structures for trees.

(Refer Slide Time: 1:31)



What is a tree? Many of you might have come across a tree before, except this tree is going to be different from one that you have seen before. The root will be at the top. In most of the trees around, you do not see the root.

(Refer Slide Time: 1:37)



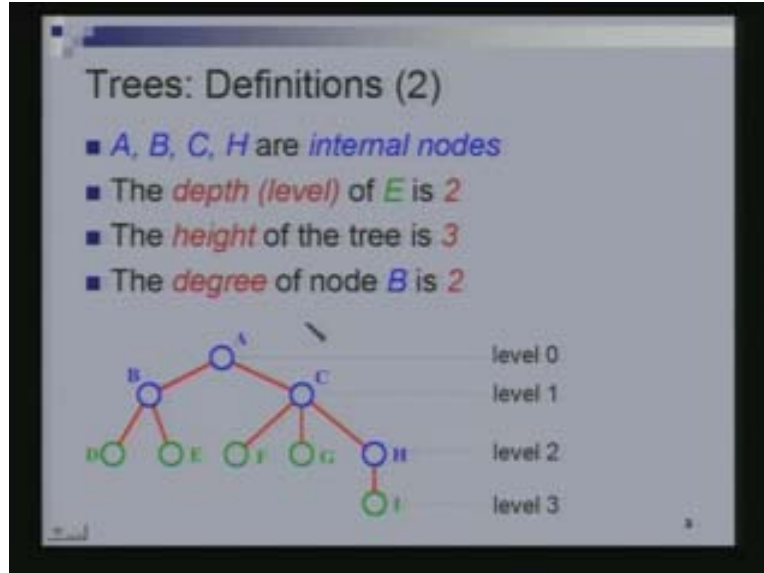
The root is going to be at the top of the tree. In the tree given in the slide above A is the root. There is a notion of parent and children, the node B is the parent of node D and E. By the same argument A is the parent of B and C, C is parent of F, G and H. A is a parent of B which in turn parent of D and E, so A is ancestor of D and E. A is also an ancestor of F, G and H. A is also an ancestor of I. A is a grandparent, sometimes we use the term grandparent. A is a grandparent of D, E, F, G and H.

Hope you understand the difference between ancestor and grandparent. D and E are descendants of A, in fact B, C, D, E, F, G, H, I are all descendants of A. C and B are siblings because they have the same parent. B is a sibling of C and C is a sibling of B. G and E are not siblings but F, G and H are siblings. D and E are the children of node B. A is a parent of B, B is a parent of D and E, D and E are children of B, B and C are children of A and all of these are descendants of A.

I have 3three ancestors H, C and A. H is the parent, C is the grandparent and A is the great grand parent but we do not use that term we just call it as an ancestor. The terms we defined till now were more in the nature of a family tree and then we will come to real trees. D, E, F, G, I are called the leaves of the tree. A is the root, if you just turn it upside down then the extremities should be the leaves.

What is the leaf? The generic term for A, B...I are also called nodes of a tree. A leaf is a node which has no children. If a node has no children then it is a leaf. H is not a leaf since H has a child but I, F, G, E and D are nodes which do not have any children and so they are called the leaves.

(Refer Slide Time: 5:08)



A, B, C and H are called internal nodes, a node which is not a leaf is called an internal node. We associate a notion of level with each node, the root is at level 0. The children of the root are at level 1. The children of those nodes which are at level 1 are at level 2. D, E, F, G and H all are at level 2. It is not that H is at level 2, all of these nodes are at level 2. I is at level 3.

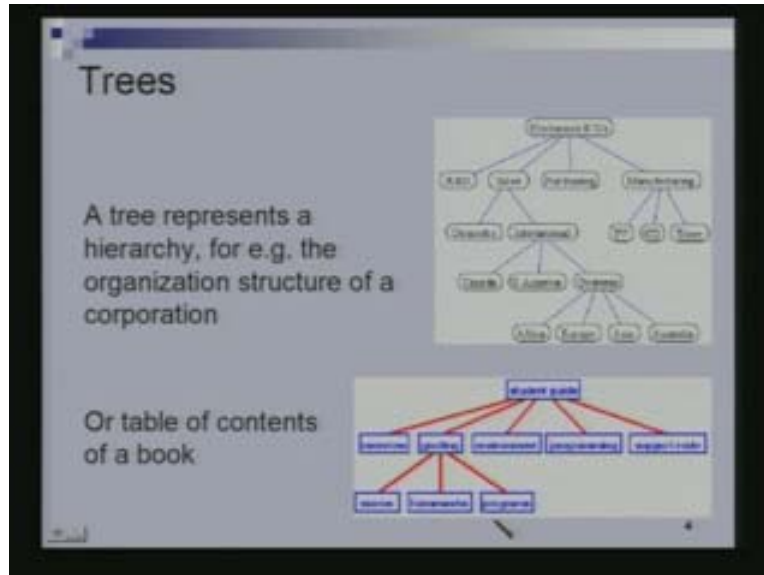
Sometimes we also use the term depth, in which depth and level are the same thing. The level 0, level 1, level 2 are also called as depth 0, depth 1 and depth 2. The height of the tree is the maximum level of any node in the tree. What is the maximum level of any node in the tree? The height of this tree is 3.

The degree of the node is the number of children it has. B has a degree 2, C has a degree 3 and H has a degree 1. The leaves have degree 0 because they do not have any children. Basic terminologies are quite intuitive. What are trees used for?

They can represent the hierarchy in an organization. For instance there is a company let us call electronics R Us which has some divisions. RND is 1 division, purchasing is another and manufacturing is 4th division. Domestic and international are the sub-division for the sales. You could represent the organizational structure through a tree. You could also use a tree to represent the table of contents in a book.

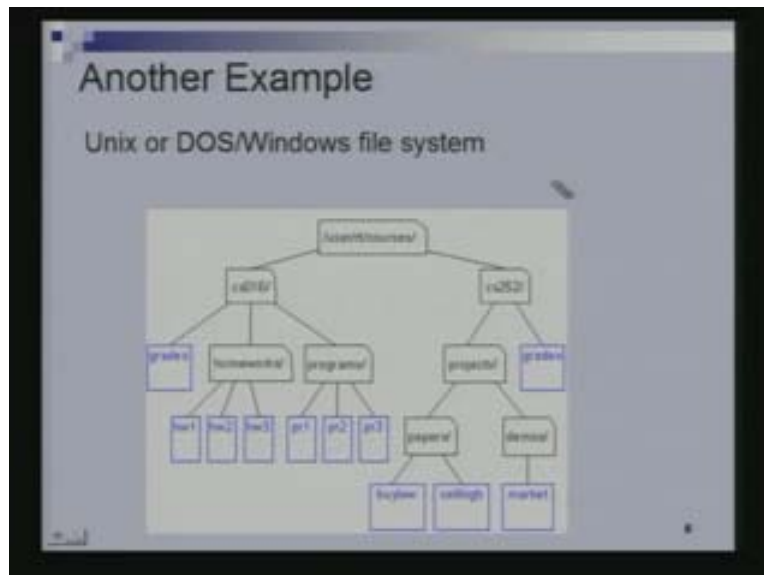
Let us say a book called student guide which has chapters on overview, grading, environment, programming and support code. The chapter grading has some sections called exams, homework and programs. They could also have some sub-sections and that would build up the tree.

(Refer Slide Time: 6:56)



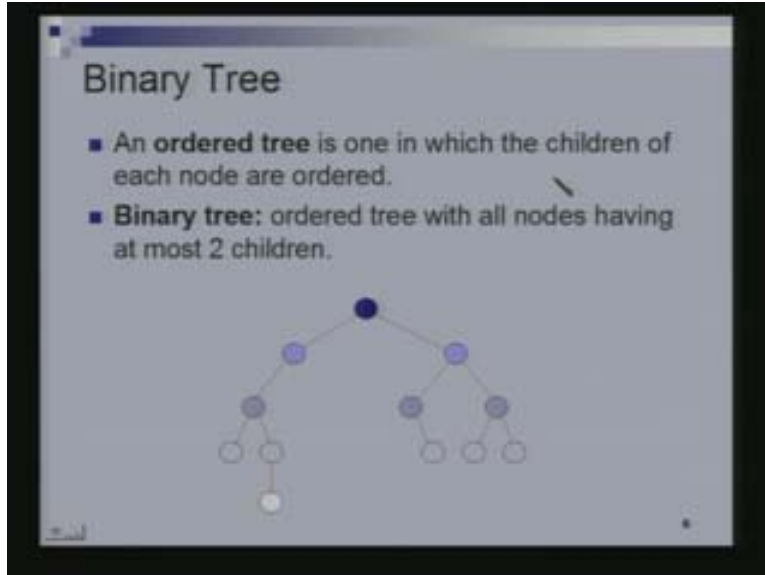
Your file system in which if you use the Unix environment or the Windows environment is also organized as a tree. The one in the level 0 is the root directory and in the 1st level are those 2 sub-directories. Then within the sub-directory I have some other sub-directories and within that I have homework, assignment and so on. Your file system is also organized like a tree.

(Refer Slide Time: 7:57)



Today in our class we are going to see about definitions and then we start using those definitions in our later classes.

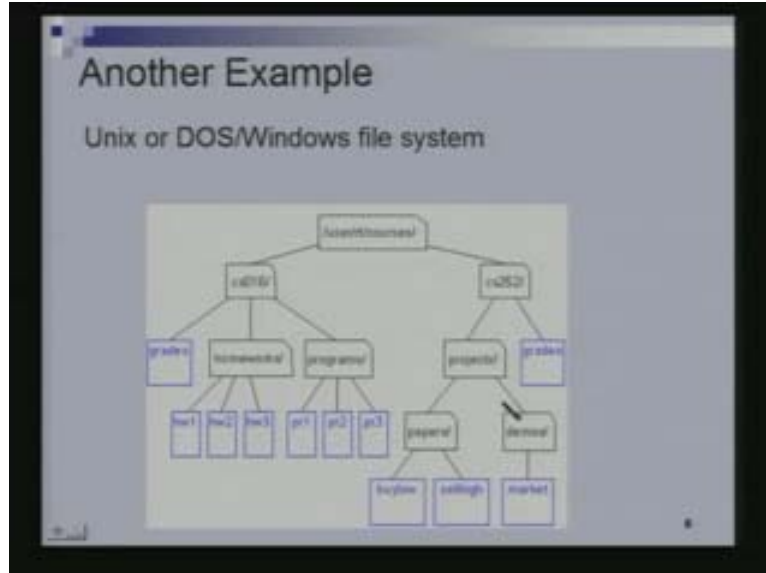
(Refer Slide Time: 8:33)



An ordered tree is one in which the children of the each node are ordered. That means there is a notion in which we would like to put the left child in the 1st level to the right side. Suppose if you want to draw a family tree, you may want to draw the eldest child to the left and the younger child as you move from left to right. There is a notion of order there and some time you want to reflect that order in your tree. But there would be no notion of order in the following example.

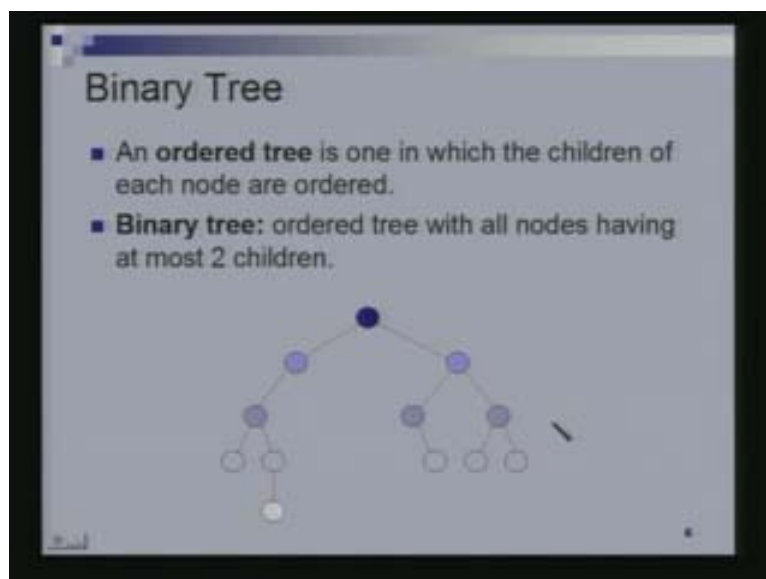
The node which is in the level 0 is a directory and in the 1st level there are two sub-directories. Whether I place the left node to the right or the right node to the left, it does not really make any difference as far as the picture is concerned. Also it does not convey any additional information but sometimes you might have the notion of order between the children. Such a tree is called an ordered tree.

(Refer Slide Time: 9:37)



A binary tree is an ordered tree in which there is a notion of left child and a right child. Actually it is an ordered tree in which every node has at most 2 children. The diagram given in the slide below is an example of a binary tree. The root node has 2 children and the child node on the left has only 1 child and the following child on the right has only 1 child. The node which does not have child node are said to be leaves. We have 5 leaves which have no children. These nodes in the 1st level are ordered and there is a notion of left and right nodes. If I were to change the tree that is if I were to draw the left nodes on the right and right nodes on the left then I get a different binary tree. That would still be a binary tree but it would be different from this binary tree.

(Refer Slide Time: 11.06)

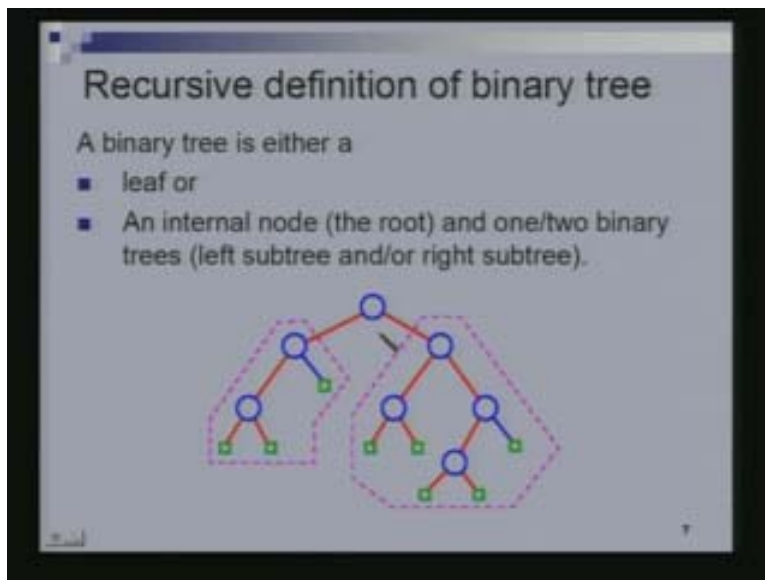


All of this is dependent upon the application you have. This is just a way of representing information. Sometimes the order has meaning to it, sometime it has no meaning to it. When it has some meaning to it then you would rather use an ordered binary tree and when you change the order, then you are representing something different. We will see more example of this and things would become clear.

I can also define a binary tree in a recursive form as follows. A binary tree is just a single node or a leaf or it is an internal node which is the root to which I have attached 2 binary trees. In the following slide the nodes which are marked on the left side are called left subtree and the nodes marked on the right are called right subtree. I can construct any binary tree in this manner.

I take a node and attach a left subtree and a right subtree. I get a left subtree and right subtree through recursive in which it is obtained by taking a node and attaching it to the left and right subtrees.

(Refer Slide Time: 12:12)



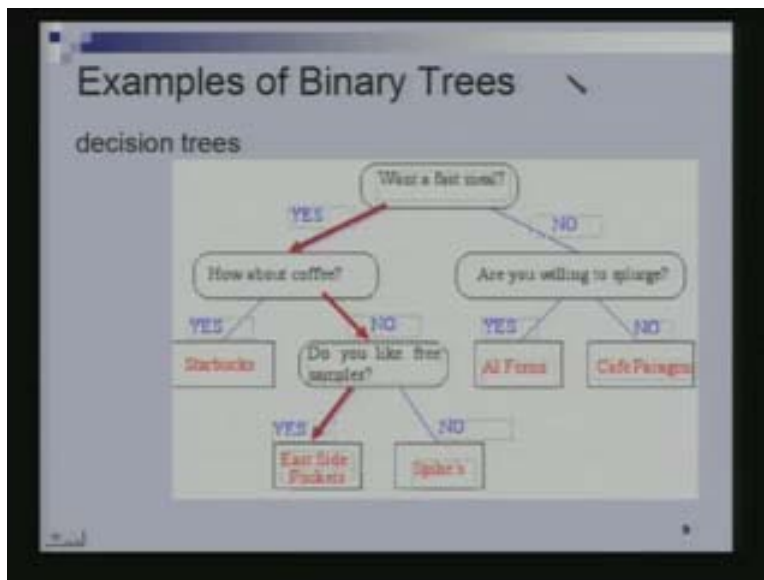
I have said and/or which means this left subtree might be null that is I might not attached anything or I might not attached anything to the right or I might have attached both the subtrees. Remember we have introduced other terms, left subtree and right subtree. The node to the left side of the root node is called the left subtree and the node on the right side is called right subtree. What is the left subtree of the node which is in the 1st level? In the 2nd level, the node at the extreme left is the left subtree of the node.

plus operator in it. Whatever is the resulting value we are adding that to 1. I draw a tree whose root is a plus operator and one child is 1 and the other child is the subtree that is obtained from this operation and I could build this tree. This is just another way of representing arithmetic expression.

Decision tree is another example of binary tree. The example given in the slide below is taken from the book. Star bucks, Café paragon and most of it would not make much sense, may be we would not come across them. What is the decision tree?

Each node in the decision tree corresponds to some decision that you want to make. You come to root node and ask whether you want a fast meal. The answer is yes then you come to the left node and whether you want coffee or not. The answer is yes then you go to star bucks. If the answer is no you may go to some other place and so on. Thus decision trees are another example of binary trees. Why because typically it is yes and no. You would follow the decision tree to get into a particular node.

(Refer Slide Time: 14:51)



This was just more of terminology and examples. Let us see more concrete stuff. Let us define a complete binary tree. We are still at binary trees, as you can see every node in this tree has less than or equal to 2 children or at most 2 children. But I will call such a tree as a complete binary tree. We call a tree as a complete binary tree if at the i^{th} level there are 2^i nodes. In some sense it is full and when every node has 2 children it does not give you a complete binary tree.

(Refer Slide Time: 16:06)

Complete Binary tree

- level i has 2^i nodes
- In a tree of height h
 - leaves are at level h
 - No. of leaves is 2^h
 - No. of internal nodes = $1+2+2^2+\dots+2^{h-1} = 2^h-1$
 - No of internal nodes = no of leaves - 1
 - Total no. of nodes is $2^{h+1}-1 = n$
- In a tree of n nodes
 - No of leaves is $(n+1)/2$
 - Height = \log_2 (no of leaves)

I will show you why it cannot be a complete binary tree. Let us look at the slide below and check whether every internal node have 2 children in this tree. Every node has a 2 children then that tree should also have leaves. It cannot be the case in which every node has 2 children, in some case there are no children. Just with the requirement that every node has 2 children, every node other than the leaf that means every internal node has 2 children does not implies it as a complete binary tree. This is a counter example to that in which every internal node has 2 children. This is not a complete binary tree.

(Refer Slide Time: 17:00)

Examples of Binary Trees

arithmetic expressions

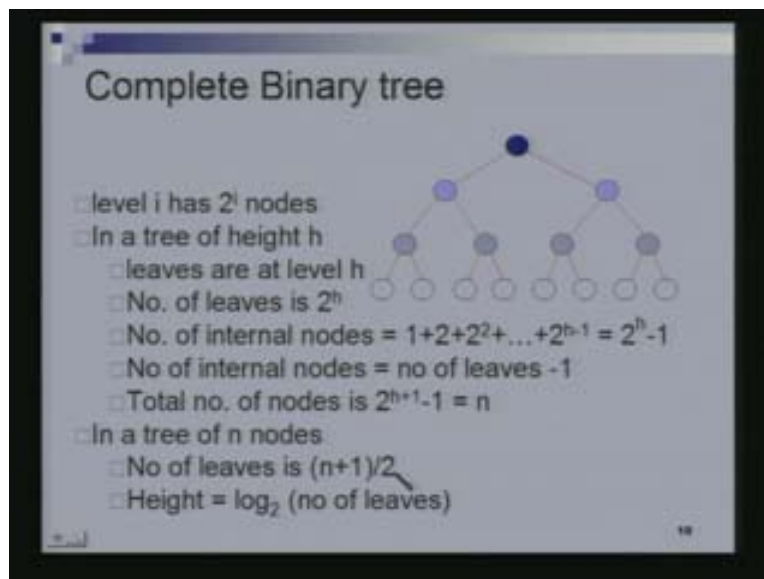
$(3 + (1 + 4 + 6)) + (5 + 4) + (7 + 2)$

The following is an example of a complete binary tree. We want to say that at level i there are 2^i nodes. The root node is at level 0 that is 1 node, at level 1 there are 2 nodes, at level 2 there are 4 and at level 3 there are 8.

If h is the height of the tree, in the following example what is the height of the tree? We call height as the maximum level number so we should not count this as 4. Thus the height of the tree is 3. If h is the height of the tree that means all the leaves are at level h then by the definition of the binary tree we have said that the level i has 2^i nodes that means there are 2^h leaves. The number of leaves in a complete binary tree of height h is just 2^h .

What is the number of internal nodes? At level 0 we have 1 node, at level 1 we have 2 nodes and so on. Thus the sum is given as $1+2+2^2+\dots+2^{h-1} = 2^h - 1$, because at level h all the nodes are leaf nodes. Thus the sum is $2^h - 1$, this is the number of internal nodes and the number of leaves is 2^h . The number of internal nodes is the number of leaves-1. This is for a complete binary tree.

(Refer Slide Time: 17:34)



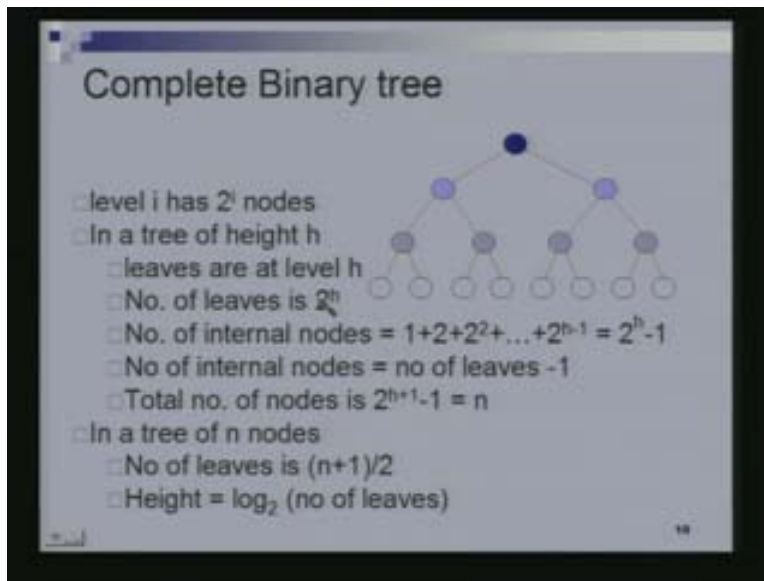
What is the total number of nodes in this tree? It is 2^h which is the number of leaves + $2^h - 1$ which is the number of internal nodes, hence it becomes $2^{h+1} - 1$. Let us call this number as n . If I have a complete binary tree of n nodes, what is the height of this tree?

Let us go one step at a time. What is the number of leaves in this tree? If the number of nodes is n and the number of leaves was 2^h which equals $\frac{n+1}{2}$, just from this ($2^{h+1} - 1 = n$)

expression. The number of leaves in a complete binary tree on n nodes is $\frac{n+1}{2}$. If I have a complete binary tree on n nodes, half of the nodes are leaves and the remaining half are

of internal nodes. Similarly I can say that if I have a tree on n nodes, then the height of the tree is $\log_2(\text{no of leaves})$. I can evaluate h from $(2^{h+1} - 1 = n)$, h will be $\log_2\left(\frac{n+1}{2}\right)$ and so it is the \log_2 (no of leaves). Else we can go directly from 2^h , where the number of leaves is 2^h and so h is \log_2 (no of leaves).

(Refer Slide Time: 20:37)

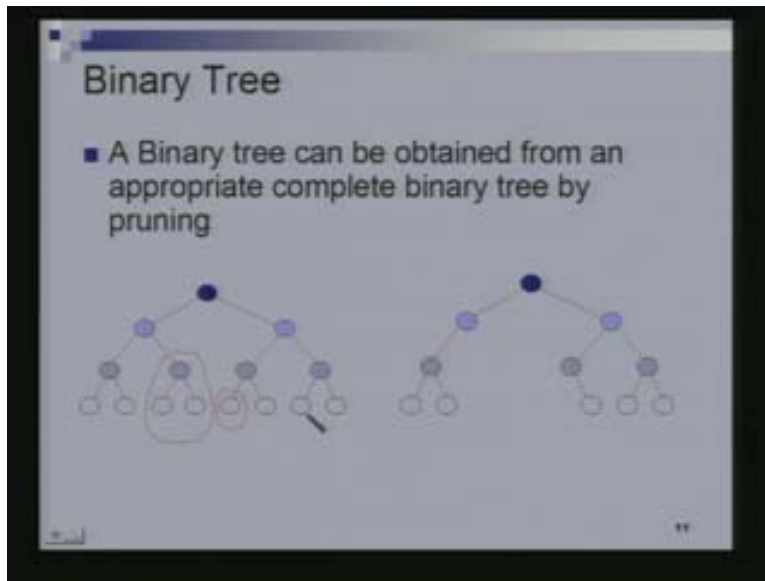


You are just doing some simple counting here. If I give you a complete binary tree of height h then you should be able to say about the number of leaves and the number of internal nodes it has. When I give you a complete binary tree on n nodes, you should be able to say the height and so on. If you have a tree on n nodes then the height of the tree is $\log_2\left(\frac{n+1}{2}\right)$.

The other thing you have to keep in mind is that in such a tree the number of leaves is very large. It is roughly half the total number of nodes. It is very leafy kind of a tree. So far we have seen a complete binary tree but a binary tree is any tree in which every node has at most 2 children. To get any binary tree, you can start with a suitably large complete binary tree and just cut it off.

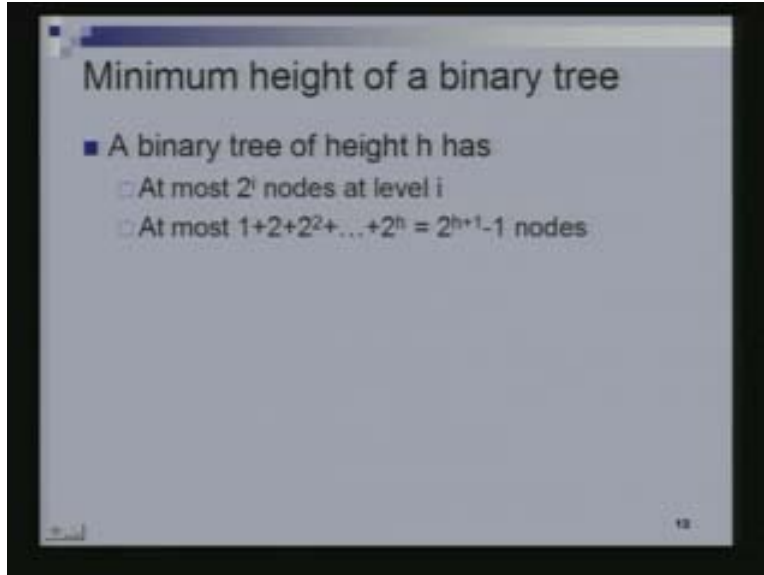
For instance if I were to cut off some pieces then I would get a binary tree as shown in the slide below. I can always do it, no matter about the tree I need. Take the binary tree on the right side as height 3 then I would start with the complete binary tree of height 3 which is on the left. Just cut off some pieces on the left side of the tree to get the tree which is on the right side. The picture given in the slide below is the proof.

(Refer Slide Time: 22:07)



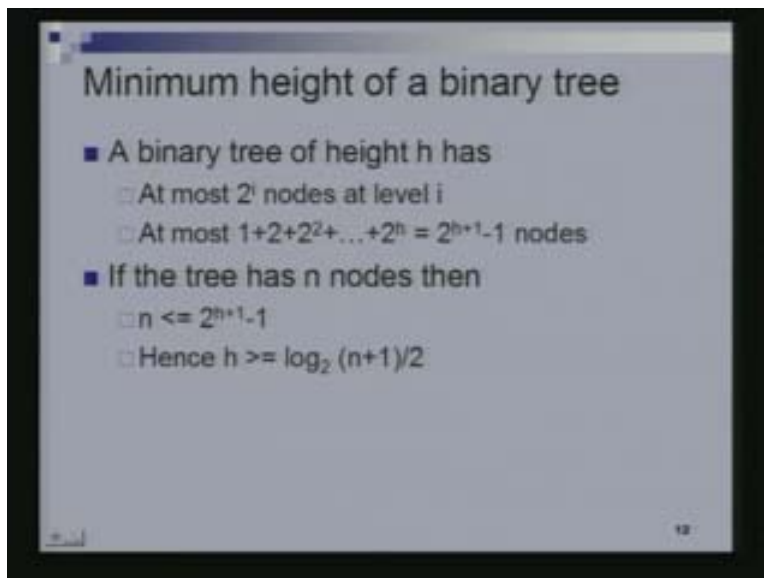
Let us use this fact that you can obtain any binary tree by just pruning of a complete binary tree. Take a complete binary tree, cut off some branches then you will get a binary tree. If I have a binary tree of height h then in a complete binary tree at level i there were atmost 2^i nodes. In a binary tree at level i there will be atmost 2^i nodes, there cannot be more than 2^i nodes because the binary tree is obtained from a complete binary tree by pruning.

(Refer Slide Time: 23:14)



This is an important fact, at most 2^i nodes at level i implies that the total number of nodes in your binary tree of height h is at most $1+2+2^2+\dots+2^{h-1} = 2^h - 1$ nodes. The last level is h , at level 0 there will be 1 node, at level 1 there is at most 2 nodes, at level 2 there are at most 4 nodes and so on. This is the maximum number of nodes that binary tree can have.

(Refer Slide Time: 24:23)



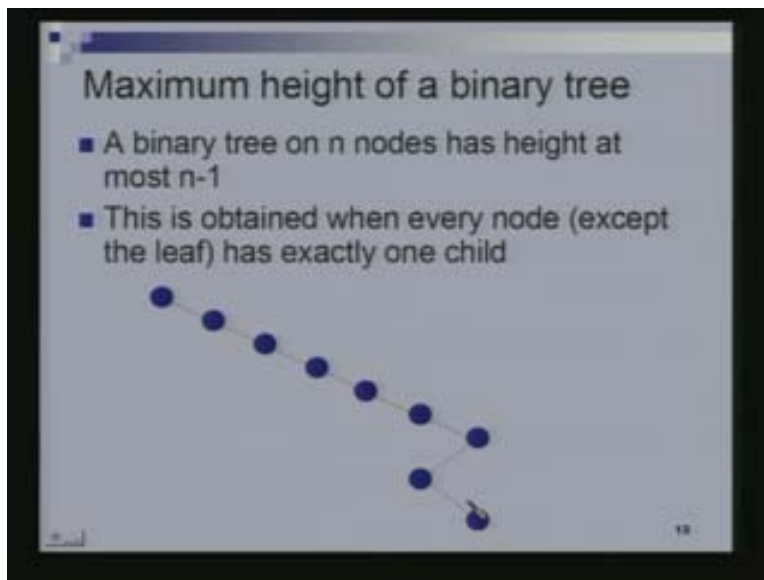
Let us rewrite this. Suppose I told you that a tree has n nodes. Then n is less than or equal to this $(2^{h+1} - 1)$ quantity, $n \leq 2^{h+1} - 1$ which means that the height of the tree is just

rearranged and it is $h \geq \log_2 \frac{(n+1)}{2}$. If I give you a binary tree with n nodes in it, its height is atleast $\log_2 \frac{(n+1)}{2}$ and there is a particular binary tree which achieves this equality and that is a complete binary tree.

Think of a complete binary tree as a tree which acquires the smallest height. If I create a binary tree with the certain number of nodes, the one which has the shortest height will be a complete binary tree. Because there we are packing all the nodes as close to the root as possible by filling up all the levels to the maximum. That is the minimum height of the binary tree. I give you a binary tree on n nodes, its minimum possible height is $\log_2 \frac{(n+1)}{2}$.

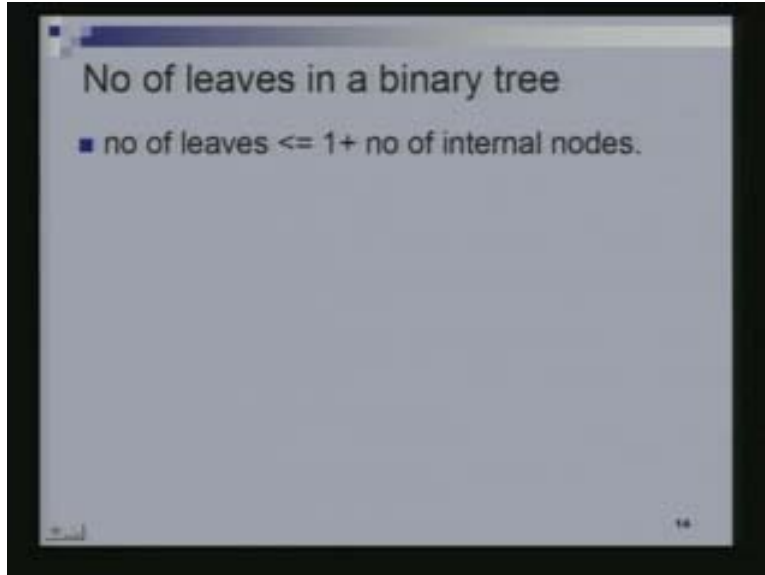
What is the maximum height that a binary tree on n nodes can have? A binary tree on n nodes has height atmost $n-1$. This is obtained when every node has exactly 1 child and the picture is given in the slide below. This would be a zigzag in any manner and the height is 8 since there are 9 nodes in it.

(Refer Slide Time: 25:35)



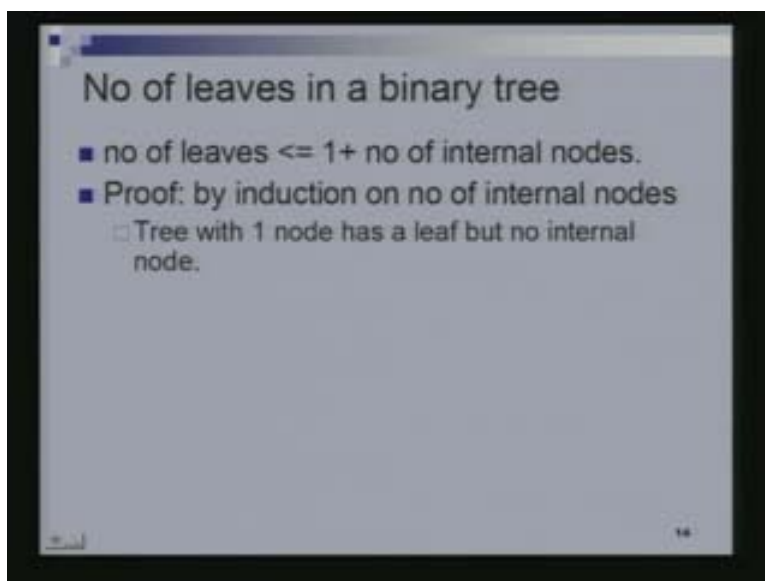
In a binary tree on n nodes the minimum height is $\log(n)$ that is $\log\left(\frac{n+1}{2}\right)$, but we say it as $\log(n)$ and the maximum height is $n-1$. That is the mistake many people make. They always assume that binary tree means height is $\log(n)$. But it is not the case, it could be anywhere between $\log n$ and n . How many leaves does the binary tree have?

(Refer Slide Time: 26:44)



What is the minimum and the maximum number of leaves it can have? Let us figure it out. We will prove that the number of leaves in a tree is $\leq 1 +$ no of internal nodes. This is the useful inequality, in any binary tree the number of leaves is $\leq 1 +$ the number of internal nodes or atmost the number of leaves in a tree can be 1 more than the number of internal nodes. How will you prove this? We will prove it by induction on the number of internal nodes.

(Refer Slide Time: 27:19)



In a base case consider a tree with 1 node. If a tree has only 1 node how many internal nodes does it have?

It is 0, because that 1 node does not have any child so that is the leaf. Base case is when the number of internal nodes is 0, in which case the right hand side is 1 that is the number of leaves is 1 so the inequality is satisfied.

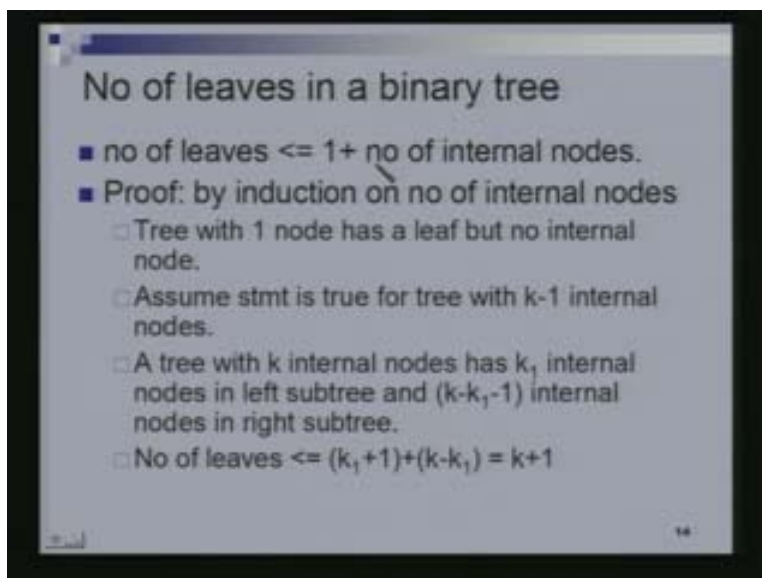
We will assume that the statement is true for all trees with less than or equal to $k-1$ internal nodes. This should be read as, the statement is true for trees with atmost $k-1$ internal nodes not just $k-1$ but anything even for less this statement is true.

We will prove it for a tree with k internal nodes. Suppose I have a tree with k internal nodes, let us say on the left subtree I have k_1 internal node. Then how many internal nodes do I have on the right subtree? It is exactly $k-k_1-1$ and not atmost because all the internal nodes are either in the left subtree or in the right subtree or it is the root node. The minus one is because of the root node. This is the number of internal nodes in the right subtree.

Let us apply the induction hypothesis. k_1 is less than or equal to $k-1$ and the quantity $(k-k_1-1)$ is also less than or equal to $k-1$. We can use the induction hypothesis. In the left subtree which has k_1 internal nodes, the number of leaves is less than or equal to k_1+1 . In the right subtree the number of leaves is less than or equal to $k-k_1-1+1$ which is $k-k_1$. The total number of leaves is just the sum of these two $((k_1+1) + (k-k_1))$, all the leaves are either in the left subtree or in the right subtree. The total number of leaves is just the sum which is $k+1$ that is we wanted to prove.

Since we started a tree with a k internal node, you have to show that the number of leaves is less than $1+k$. This is a simple proof which shows the number of leaves is atmost $1+$ the number of internal nodes.

(Refer Slide Time: 27:44)



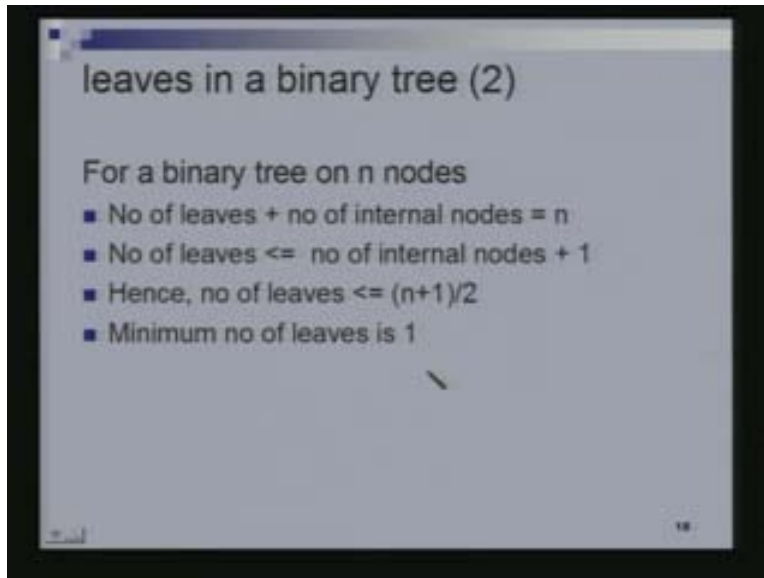
No of leaves in a binary tree

- no of leaves $\leq 1 +$ no of internal nodes.
- Proof: by induction on no of internal nodes
 - Tree with 1 node has a leaf but no internal node.
 - Assume stmt is true for tree with $k-1$ internal nodes.
 - A tree with k internal nodes has k_1 internal nodes in left subtree and $(k-k_1-1)$ internal nodes in right subtree.
 - No of leaves $\leq (k_1+1)+(k-k_1) = k+1$

14

There was a tree in which we saw the number of leaves is equal to the number of internal nodes +1. It was in a complete binary tree. What was the number of leaves in a complete binary tree? The number of leaves was 2^h , if h was the height of the tree and the number of internal nodes was $1+2+2^2 + \dots + 2^{h-1} = 2^h - 1$ nodes. There was exactly a difference of 1 between the number of leaves and the number of internal nodes. The complete binary tree once again achieves the equality. For any other tree the number of leaves will only be less than or equal to this sum. How small it can be?

(Refer Slide Time: 30:53)

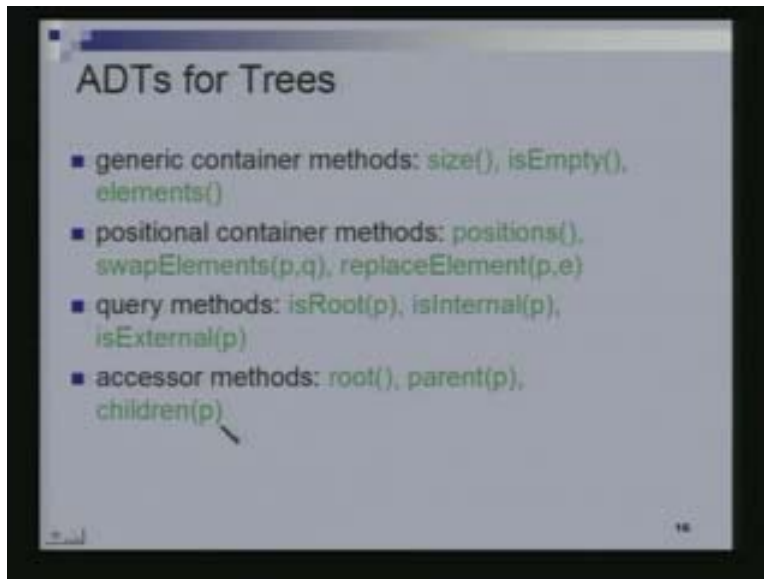


Let us look at that. For a binary tree on n nodes, the number of leaves + the number of internal nodes is n. Because every node is either a leaf or an internal node. Also we just saw that the number of leaves is less than or equal to the number of internal nodes +1. I will just rearrange, this implies that the number of leaves is $\leq \frac{(n+1)}{2}$. I have just rearranged, as the number of internal nodes is greater than or equal to the number of leaves -1. I replace that and get the number of leaves as $\leq \frac{(n+1)}{2}$ for any binary tree.

The another thing to keep in mind is for any binary tree the number of leaves will never be more than half the number of nodes in the tree. Again this equality ($\leq \frac{(n+1)}{2}$) was achieved for our complete binary tree, which is the most leafy tree. All others trees are dry and the minimum number of leaves that tree might have is just 1. The example for that is the same example that I have showed you before. The tree on 9 nodes has only 1leaf in it.

Let us look at an abstract data type for trees. You would have the generic methods which you seen for all the abstract data types till now. The following are the generic container methods, size () which tell us about how many nodes are there in the tree, isEmpty () tells whether the tree is empty or not and the method elements () which list out all the elements of the tree.

(Refer Slide Time: 32:22)



You can have position based container methods, it is as the kind we saw for the list or sequence data types. The swapElements (p, q) in which I have specified 2 positions p and q. Think of the positions as references in to the tree except that using the position data type I am not able to access anything else but the elements sitting at that position.

The method positions () will specify all the positions in the tree. It will give you all the positions in the tree as a sequence. The positions method has no parameters, when you invoke it on a certain tree it will just give you a sequence of all the positions in the tree, references to all the nodes in the tree. Once you access a particular position then using the element method on that positions you can access the element in that node.

The swapElements (p, q) given 2 positions p and q, you are swapping the elements at these 2 positions. replaceElement (p, e) which means that given a position p you are replacing the element at that position with e. In query methods given a particular position isRoot (P) is this the root of the tree. Given a particular position isInternal (p) is this is an internal node, given a particular position isExternal (p) is this external or leaf. Sometimes we use external or sometimes we use leaf, does this position correspond to leaf.

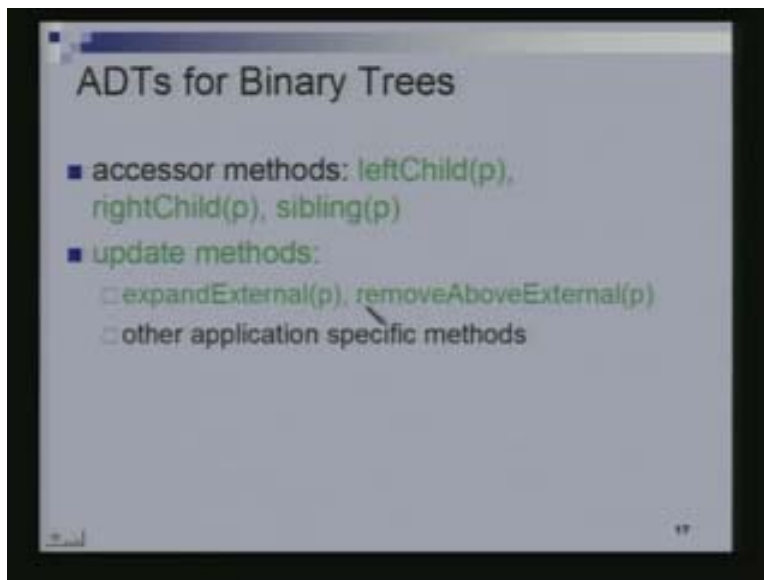
In accessor method when I call root it will return a position of the root, an object of type position. isRoot (p) is determined as given a position is it a root and root () returns the position of the root. Hope you understand the difference between the both. The position of the root means it is not a reference to that particular node but it is a reference of type

position so that you cannot access anything except the element. This was the same as the type casting which we did earlier. The method parent (p), given a particular position returns the parent node. The children (p), given a particular position returns the children of this node. If it were children, there could be of many children for a certain node.

How it will return the various children? It will return as a sequence, it will return a sequence of object type sequence which will contain the position of all the children. Position has an element method which will let you to access the data. The update methods are typically application specific and this would be the generic method for a tree.

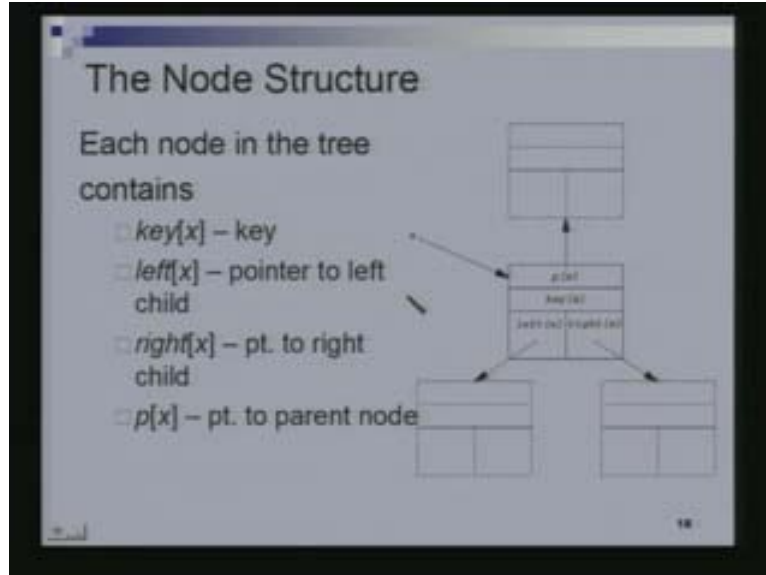
Binary tree should really be treated as a sub-class, as a derived class from a tree. All we need to do is to continue to have the same method as we described for the tree but we will have some additional methods. There would be a notion of a left child given a position give me the left child, give me the right child or give me a sibling.

(Refer Slide Time: 36:37)



We will come to the update methods when we see the example of it. What is the node structure in a binary tree? What are the kinds of data that you would be keeping in an object corresponding to a node of the binary tree?

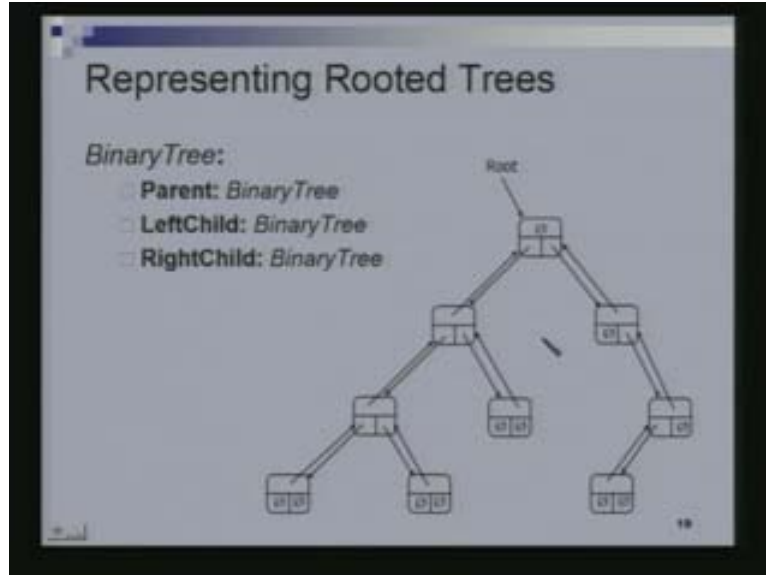
(Refer Slide Time: 37:14)



You would have the data, you would have a reference to the left child and a reference to the right child and you would have reference to the parent typically. You would also have a reference to key or data associated with this node, any element that is sitting in this node you would have reference to it and these were all sitting together. The reference to the node which is at the center will not be stored in the node and that does not make any sense.

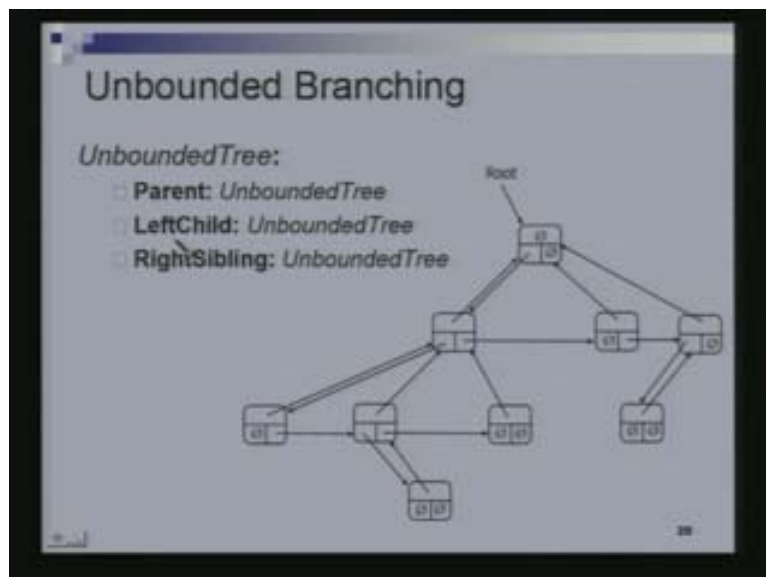
For instance if I access to this node, suppose this was the root node and I use the root method to get a position to this node then using that position I can now access the left child by invoking the left child method and in this manner I can get the position of any node. Once I have the position of a node I can then invoke element method and any method to get the data associated with that. A node in this case would definitely implement the position interface. In the slide given below, this is what the binary tree would look like if you look at the links and so on.

(Refer Slide Time: 39:12)



The parent link would be null for a root node because it has no parent. Then it would have left child in which the left child would be referring to the node on the left and the right child would be referring to the node which is on the right and so on. In the above diagram the extreme right node does not have any right child, its right child member would be referring to null. That was for a binary tree. How do we take care of arbitrary trees? Let us say unbounded trees. The root node has 3 children and its child which is on the left also has 3 children.

(Refer Slide Time: 38:39)



Are we going to have 3 different data members to refer to 3 children? That is not clear about how to do it, because then if it has 4 children then how you would create space for another member. The way to do it is that you have a reference to 1 of the child only and then all the children are in a linked list. Each child will have a reference to the parent, so all of these children would be pointing to the parent node. But the parent node would be pointing to only 1 of them.

Which would be the head of the linked list in the 2nd level? From the node which is at the 1st level, if I want to refer to the children, I can just come here essentially return all the elements of this linked list. How do I know that I have reached the last element of the linked list when the next is empty?

The 1st field of the node is empty because it does not have children. Every node still has only 3 data members, parent or 3 references 1 for the parent, 1 for left most child and 1 for the right sibling. The left node on the level 1 would refer to left most child and not to all the children because that we do not know how many are there. It will have 1 more to refer to the right sibling because for the left node at the level 2 should refer to the right sibling.

You can do with only 3 references. The node in the level 2 has only 1 child and it is just pointing to that 1 child, there is no sense of left and right here. This is not a binary tree, left and right makes sense only in a binary tree.

Actually I should not have written left child, I should have written 1st child. That is any 1 child then it just point to that 1 child and that let you to access its siblings through a linked list. From the 1st child you will go to the next child and to the next child and so on. You can step through all the various children throughout linked list. With that we will end our discussion on binary trees today. In the next class we are going to look at reversals of trees.