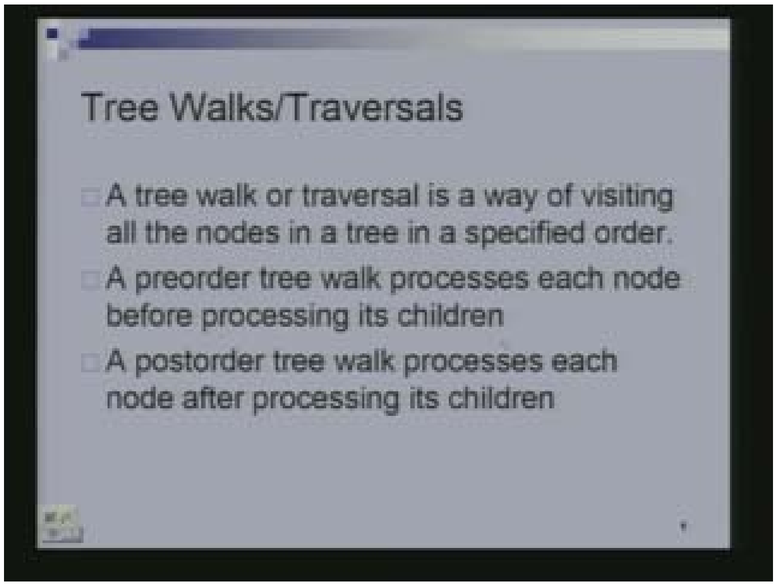
Data Structures and Algorithms
Dr. Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture – 7

Tree Walks / Traversals

In the last class we looked at definitions of trees, a binary tree and a complete binary tree and the height of these things. Today we are going to continue with our discussion on trees in particular we are going to talk about tree walk or tree traversals.

A tree walk is a way of visiting the nodes of a tree in a specified order. There are 2 different tree walks that we will consider, one is called the preorder walk and the other is called postorder walk. In a preorder walk you first visit or process each node and then you go and process its children. I will show you an example to follow this and then you will be clear.

(Refer Slide Time: 01:23)



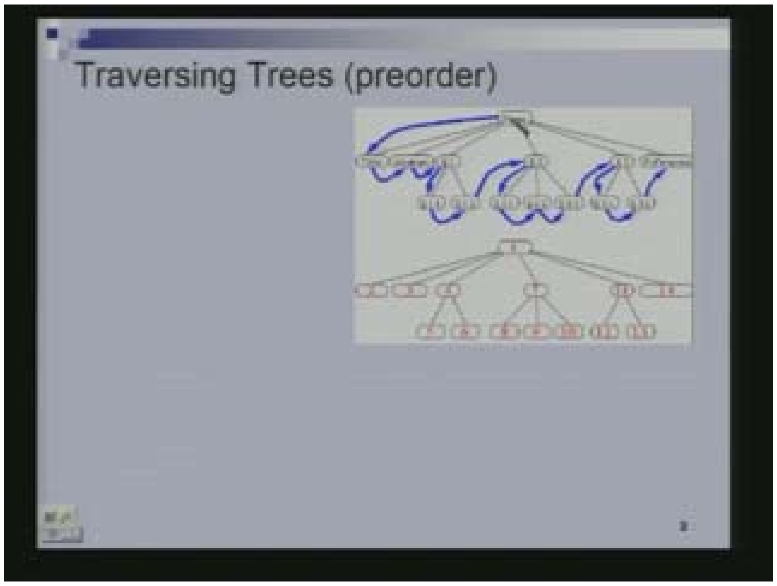
Tree Walks/Traversals

- A tree walk or traversal is a way of visiting all the nodes in a tree in a specified order.
- A preorder tree walk processes each node before processing its children
- A postorder tree walk processes each node after processing its children

In a post order you will first process all the children or visit all the children and after that only you would process the node. Let us see an example which will clarify this. I am looking at the examples of preorder tree walks.

Suppose recall that we said, we can construct a tree out of a book or a paper. We can look at the organization of a book as a tree. Let us say a research paper. You have the paper and it has a certain sections, the first section is the title, second section is the abstract which discusses about what is their in the paper and then you have section 1, section 2, section 3 and then at the end of it you have references that is what are the books and the other papers that this particular publication has referenced.

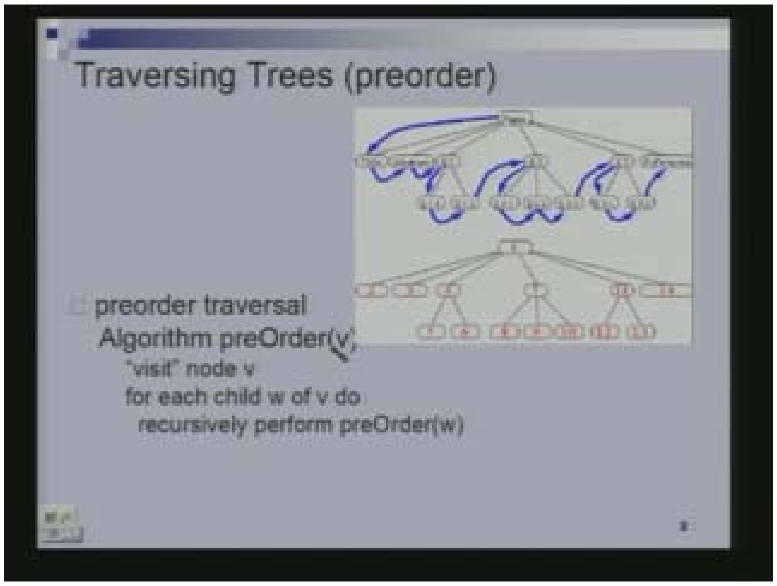
(Refer Slide Time: 02:37)



Traversing Trees (preorder)

The section 1 has 2 sub sections that is section 1.1 and 1.2. The section 2 has 3 sub sections, section 3 has 2 sub sections and so on. When you read the paper, this is the order you would go in. Suppose you are reading the paper end to end, first you will go to the title, then read the abstract then you will look at section 1 and then its sub section 1.1, 1.2 and so on. If you were to think of a book and how the table of contents of the book are listed. The way the tables of contents are listed is that first you have the chapter and then the sections within the chapter are listed. Then the next chapter and the sub sections within that chapter and so on.

(Refer Slide Time: 04:04)



Traversing Trees (preorder)

- preorder traversal

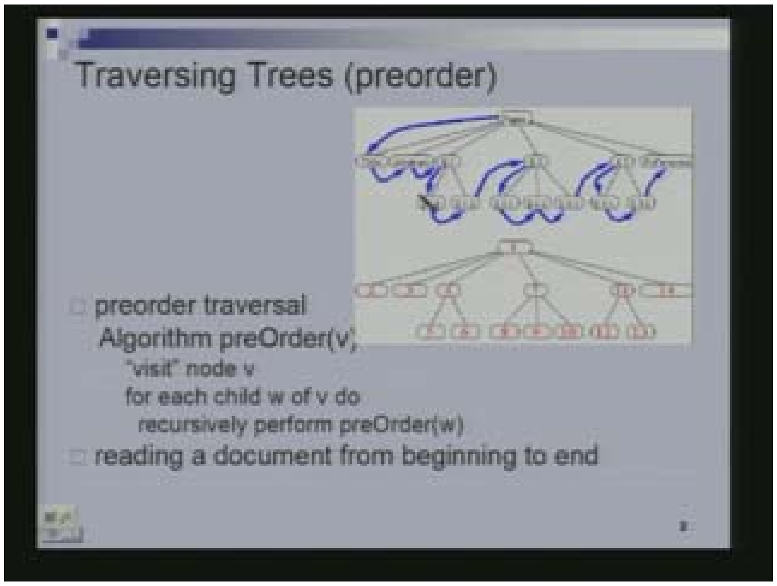
```
Algorithm preOrder(v)
  "visit" node v
  for each child w of v do
    recursively perform preOrder(w)
```

If I were to look at the above slide as the tree and these as the nodes of a tree. The first node that we are referring or accessing is the node 1. Then we go to the node 2 then 3, 4, 5, 6 and after we are done with 6, in a table of contents you will have 7, 8,9,10,11,12,13 and then 14. This is also called the preorder traversal of a tree.

The pseudo-code for preorder traversal would look something like the one which is given in the above slide. If I have to do a preorder traversal of a node v in a tree, so to begin with I would call preorder traversal at the root of the tree. Then I would say first visit the node, visit here is a generic term and we will use it very often. All it means is that I am doing some computations in that node. In this particular case if I were listing out the book as table of contents then visit would correspond to print the title or print the heading of that particular node.

For instance each node corresponds to a section or some thing then this ("visit" node v) would correspond to print out the name of the section. Then once you done that then you go to each of the children nodes and repeat the process. Repeat this same process there on each of the child nodes. Because this tree could be arbitrarily d and the subsection 1.1 could be as 3 sub sections 1.1.1, 1.1.2 and so on.

(Refer Slide Time: 04:52)



Traversing Trees (preorder)

- preorder traversal

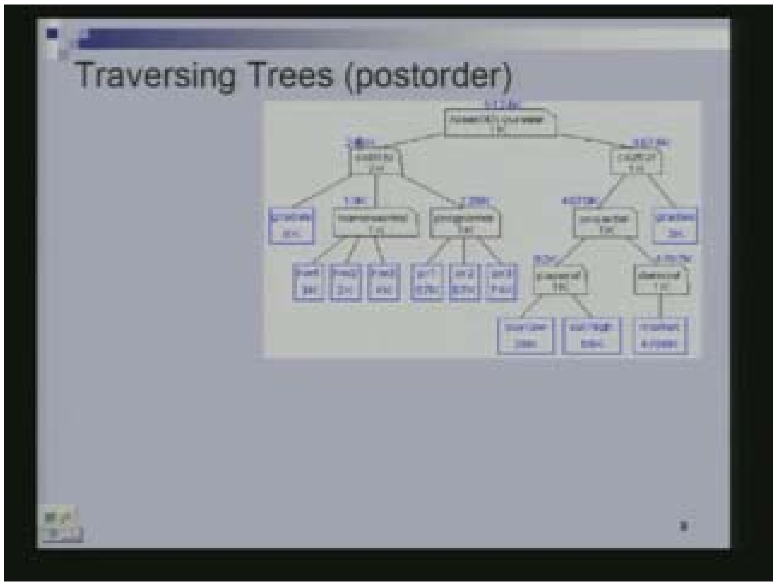
```
Algorithm preOrder(v)
    "visit" node v
    for each child w of v do
        recursively perform preOrder(w)
```

- reading a document from beginning to end

If you were to do a preorder traversal then you would come to 1, then you would come to 1.1 then you would go to 1.1.1 and then 1.1.2, 1.1.3 and only then you would go to 1.2. That is the need for this kind of a recursive traversal. What you doing here? You first visit the node then visit all its children. So these are the 6 children of that node. We are saying visit the first node. What does visit correspond to? It does not have any children so it just means visit the first node. The second one corresponds to visit that node. The third one corresponds to visit that node and then visit its children. That is what we are doing, visit its children and then the next child and so on. This is what would be called a preorder traversal.

I gave you the example this is like reading a document from beginning to end. We could also have what is called a post order traversal. In a post order traversal recall I said that we are going to visit the node at the end. We will first visit its children and only then we would visit the node.

(Refer Slide Time: 06:43)



Traversing Trees (postorder)

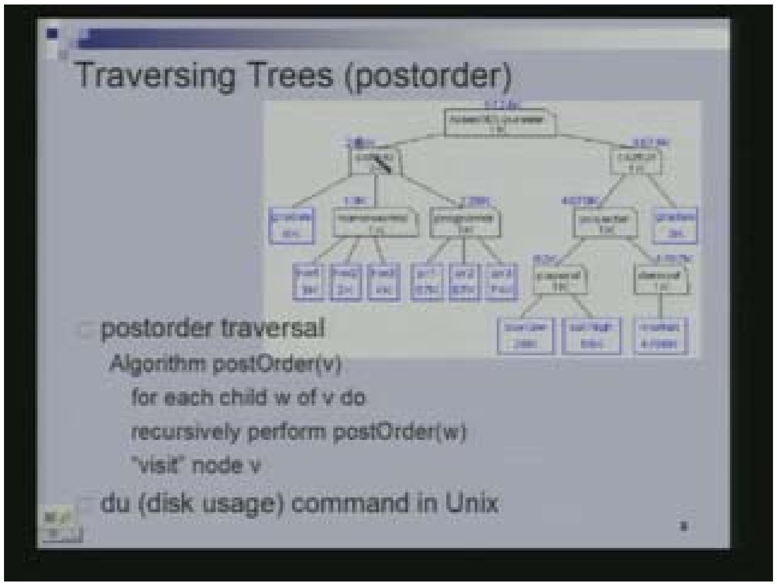
Let us say I have a directory structure like this. This is my root directory courses. This is an example from the book. There are 2 courses here CS 016 and CS 252. And then in that there are 2 sub directories, there is a file called grades. Within this sub directory there are 3 files, within this sub directory there are 3 files and so on. This is the directory structure.

Suppose I want to compute the total space occupied by this file system or this entire directory. What would I do? I would compute the total space occupied by the subdirectory which is on the left and I would compute the total space occupied by the subdirectory which is on the right and then I would add these two up at the node which is at the center to obtain the total spaces required. So in some sense I am actually visiting the node which is at the center or doing some computation on this node after having done the computation at the 2 children nodes.

After having computed the total spaces required by the sub directory on the left and after having computed the total spaces required by the subdirectory on the right, only then I am doing the node at the center. When you are doing a post order traversal of a node v, for every child of the node first we are going to perform a post order on that. In this example post order corresponds to finding the total space occupied by that sub directory. So to compute the total space occupied by the directory which is at the center, we are first going to compute the total space occupied by the sub directory on the left then the total space occupied by the directory on the right and having computed that, you are then going to compute the total space required by the directory at the center.

In some sense the order in which computation is done is reverse from the pervious example. In fact this is the order in which the disk usage command in Unix, if you ever have used this particular command in Unix. What it does is, if you type in this command in a directory it tells you what is the total space occupied by the various subdirectory their. The way you list it out is, if you were to type the disk usage command in this sub directory, it was first going to list out the total space occupied in this directory on the left then the total space occupied in this directory on the right and then eventually at the end it list out the total space required here. Because it would have computed it only after it had done the computation on the left and right. How does this do the computation? In a recursive manner that is to compute the total space required by this directory(Refer Slide Time: 09:41), first it is going to compute the total space required here then the total space required here and then add them up to get the total space required here. So that would be a post order traversal.

(Refer Slide Time: 07:05)



Traversing Trees (postorder)

- postorder traversal

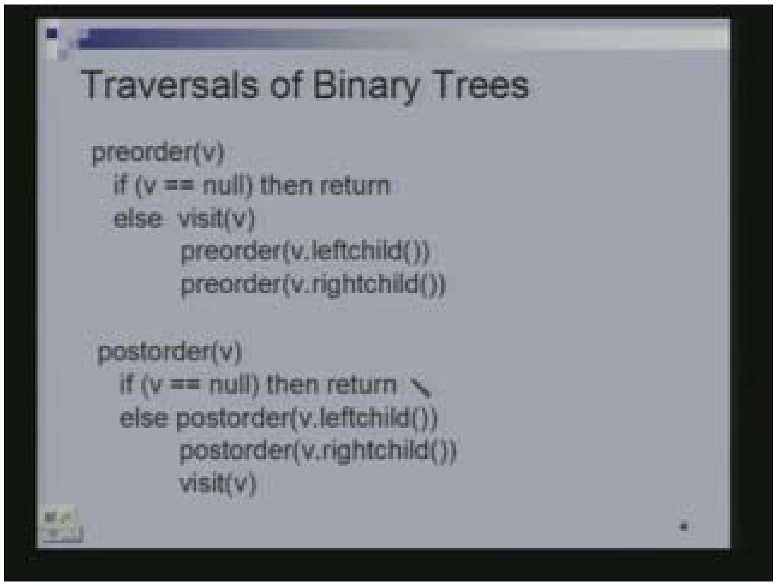
```
Algorithm postOrder(v)
    for each child w of v do
    recursively perform postOrder(w)
    "visit" node v
```

- du (disk usage) command in Unix

Which child we would visit it first? We are looking at ordered trees. There is a notion of a first child, second child, the third child and fourth child. So the first child is visited first and next is the second and so on. If you have drawn the tree in such a manner such that the first child is at the leftmost then we would say that the leftmost child is visited first and after that the other one on the right and so on. It depends upon how you have drawn your trees. What I had just shown you was traversal in general trees. If it is a preorder, first visit the node then visit the children nodes. In postorder first visit the children node and then visit the nodes.

Let us look at how this specializes to the case of binary tree. In a preorder traversal what are we saying? So v is a node, if v is null then there is nothing to be done. If v is not null then in a preorder traversal we are first saying visit v. The visit is some generic computation we do not know what it really is. It depends upon what your particular application. First say visit, then do a pre order traversal on the left child and then do a pre order traversal on the right child. Note that this is a recursive procedure. We are calling preorder within the procedure itself. How does this (preorder (v.leftchild ())) work? This will work by making the call to itself, when we are doing a preorder traversal on the left child and on the right child. And the difference between pre order and post order here is that visit now comes at the very end. First you do a postorder traversal on the left child then you do a postorder traversal on the right child and then eventually you do visit v.

(Refer Slide Time: 10:54)

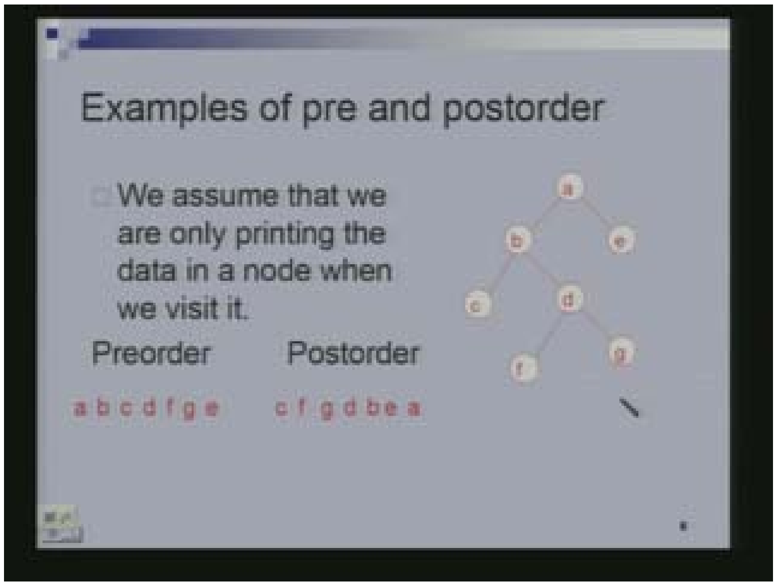


Traversals of Binary Trees

```
preorder(v)
  if (v == null) then return
  else  visit(v)
        preorder(v.leftchild())
        preorder(v.rightchild())

postorder(v)
  if (v == null) then return
  else postorder(v.leftchild())
       postorder(v.rightchild())
       visit(v)
```

Let us see if you understood this. Let us look at an example. This is my tree. It is a binary tree. I want you to tell me what is the pre order and post order traversal of this tree. What we are doing when you visit a node is that we are just printing out the contents of the node. So let us first look at pre order. What do you think would be the first thing that would get printed if we are doing a pre order traversal of a tree? The a is the root. So we come here and we print a. Then we have to go and do a pre order traversal of a left sub tree. When we do a pre order traversal of the left sub tree we are going to come to the root of the left sub tree and first visit the node, visit corresponds to printing the content. We will just print it out as b. Then we will go to the left sub tree which is c. We will come here, we will look at the root node, first visit the node. Visit the node here means printing the contents, we will print c. Then we try to go to its left sub tree but its left sub tree is null. There is nothing there. So then we go to its right sub tree which is also null nothing to be done. Now we are done with the preorder traversal of this c.

(Refer Slide Time: 11:57)



Examples of pre and postorder

- We assume that we are only printing the data in a node when we visit it.

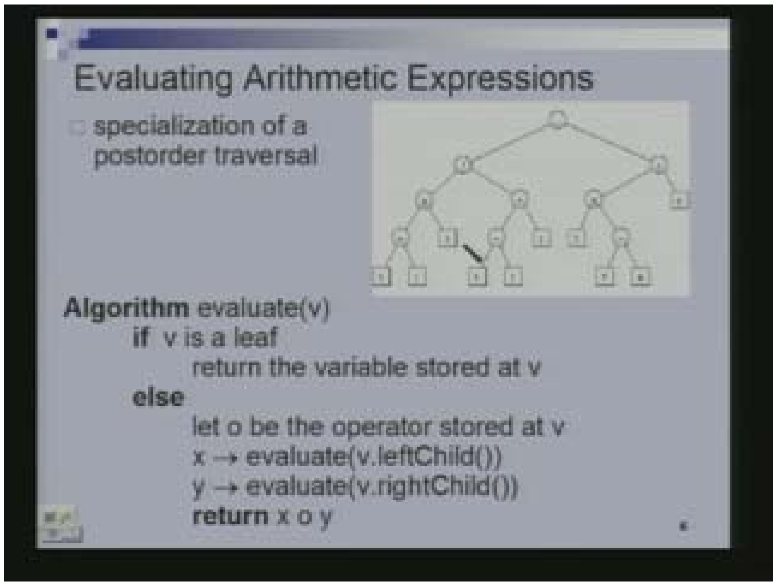
Preorder	Postorder
a b c d f g e	c f g d b e a

Where do we go now? To the right sub tree because first we went to visit the node then we did a pre order traversal of the left sub tree. Now we have to do a pre order traversal of right sub tree which means first visit the node here which is d then we go left which is f. Then again we try to go to left which is null, then go right which is also null, nothing to be done. Then we go to g and now we are done with the pre order traversal of this sub tree. We are done with a pre order traversal of the left sub tree. We are done with a pre order traversal of the right sub tree which means we are done with the pre order traversal of the entire sub tree. Now we would go to the right sub tree, the right sub tree has only e in it. So we would just print e. This would be the pre order traversal of this tree.

Let us do a post order traversal. Which do you think is the first node that would get printed? It is c. Why c the right answer? Let us see. We come here to do a post order traversal. This a will be printed at the very end after I have done the post order traversal on the left side and post order traversal on the right. So I will first try to do a post order traversal of the nodes on the left side. When I try to do a post order traversal to b, I come here, first I will do a post order traversal of c then I will do a post order traversal of d, and then print the node b. I have to come to do a post order traversal of this node c. For doing that I will first do a post order traversal of its left child which is null, nothing to be done. I do a post order traversal of its right child, null nothing to be done. So I am ready to print the content of this node c. The first thing that will get printed is c. I am done with the post order traversal on the left subtree and now I come and do the post order traversal of the right sub tree.

To do a post order traversal of the right sub tree I once again come to the root which is d. I first do the post order traversal of this left sub tree f then post order traversal of this right sub tree g and then print this content d. So post order traversal of f is a single node so it will just be f. The post order traversal of this right subtree would be g and then I would print the content of this which means d. What we will print now? So we have done the post order traversal of f and we have done the post order traversal of c so we can now print this node b. So we will print b. We have done with the post order traversal of the nodes on the left side so we go to the right sub tree e. Do the post order traversal here which means just print e and then we have done with the post order traversal here (Refer Slide Time: 15:45), we have done with the post order traversal here so we can now print the root which is a. So this would be the post order traversal. Is this clear to everyone, how the procedure works.

(Refer Slide Time: 15:59)



Evaluating Arithmetic Expressions

- specialization of a postorder traversal

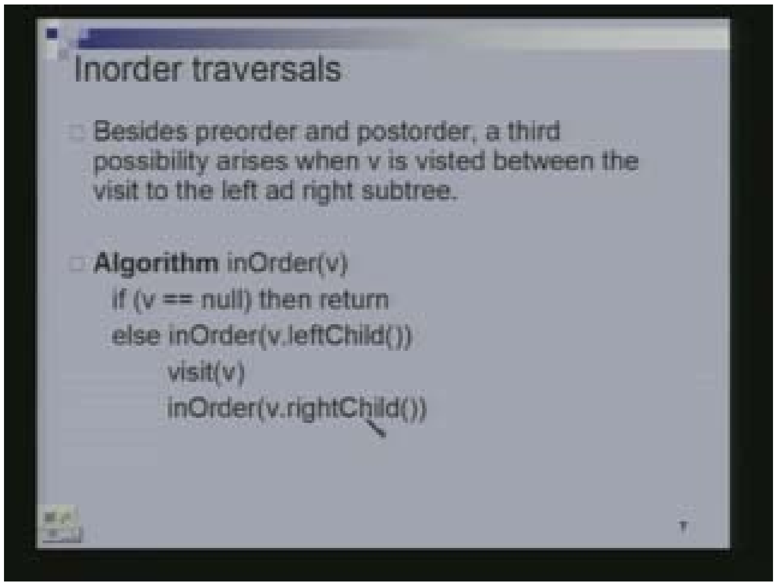
```
Algorithm evaluate(v)
    if v is a leaf
        return the variable stored at v
    else
        let o be the operator stored at v
        x → evaluate(v.leftChild())
        y → evaluate(v.rightChild())
        return x o y
```

I am showing another example for evaluating arithmetic expression. This is an arithmetic expression, we want to evaluate this expression. So how does one evaluate the expressions? This is minus, so in essence we have to compute the value of this quantity. What is the value of this sub expression? This corresponds to a sub expression so you have to compute the value of this sub expression which is on the left (Refer Slide Time: 16:36). We have to compute what the value of this sub expression on the right in the above slide. Whatever values we get we have to then take their difference. That will be the value of the entire thing. So as you can see it is like a post order traversal. First we have to compute the value of this which is on the left then the value of this on the right and then take the difference which is the operator sitting in this node at the center.

How do I compute the value on the left side of the tree? I have to compute the value of this left sub tree, I have to compute the value of this right sub tree and then do the division because that is the operator sitting here**.** We can right a procedure something like this. Suppose I say evaluate the expression corresponding to v which is a node. Let say v is a root node here and I say evaluate this. If v is a leaf then I just return the variable stored at v because that is the value. The leaf corresponds to numbers in this expression. Else if v is not a leaf then that means we are at some internal node. So to evaluate the expression corresponding to this node v, I have to first evaluate the left part. Let say evaluate (v.leftChild ()). The arrow after x in the slide should be in the other direction so that x gets the value of that. The y gets the value of the right child when I evaluate on the right child. And if o is the operator then I just compute x o y whatever that operator o is and return that value. That will be the value of expression corresponding to node v.

This is pseudo code, I hope you understand what I am trying to say here. This is like a post order traversal with a small modification. We are not going to be addressing that problem (Refer Slide Time: 18:29). The problem of generating this tree, given an arithmetic expression you will have to incorporate the priority rules to generate such a tree. We will not be worried about that for now. We are just looking at traversals. Given such a tree how can you evaluate the expression corresponding to this tree?

(Refer Slide Time: 18:53)



Inorder traversals

- Besides preorder and postorder, a third possibility arises when v is visted between the visit to the left ad right subtree.
- **Algorithm** inOrder(v)

```
  if (v == null) then return
  else inOrder(v.leftChild())
       visit(v)
       inOrder(v.rightChild())
```

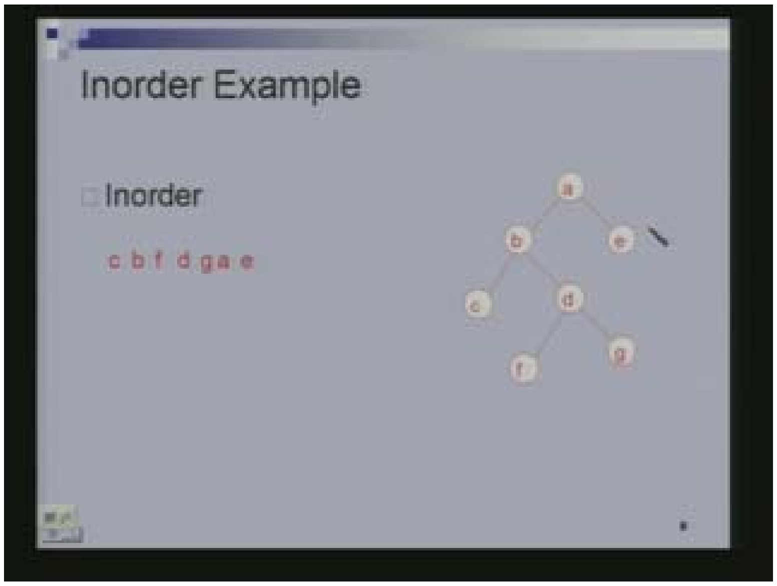
For a binary tree we have seen a preorder traversal and a post order traversal. There is a third kind of traversal which is called an in order traversal. So recall that in a pre order traversal we visited the node first, then we went to the left then we went to the right. In a post order traversal we first went to the left then to the right then we visited the node. So the third possibility is if we just visit the node, between the visits to the left and the right. There should be an and between the visit to the left and right sub tree.

So pseudo code for in order traversal would be such thing like the following. If v is null then we just get out, else we first do an in order on the left child then we visit the node and then we do an in order traversal on the right child. These are the only 3 possibilities. These are 3 binary trees, so you first go left. Where do you visit the node? Either you visit it before you visit both the left and the right or you visit it after you visit both the left and the right or you visit it in between the visits to the left and the right. These are the 3 possibilities and these are the 3 traversals that are known.

Let us just look at an example and see that we have understood inorder traversal. Which is the first node that will get printed? Is that a? First to do an inorder traversal we come to the node a. We first do an inorder traversal of this left sub tree then we come to the node a. Then we do an inorder traversal of the right sub tree e. So to do an inorder traversal of the left sub tree, we will come to b. First we will do an inorder traversal of the left then we will print the content then go right.

So to do an inorder traversal of the node c which is a single node, so it corresponds to printing c out. That is the first thing we will print. Then we will be printing the content of this which will be b and then we will have to do an inorder traversal of this right sub tree. For a right sub tree inorder traversal, once again we will first print out f then d and then g. Now we are done with the entire inorder traversal of left sub tree, so we will print out a and eventually we will print out e. That would be the inorder traversal of this tree.

(Refer Slide Time: 20:07)

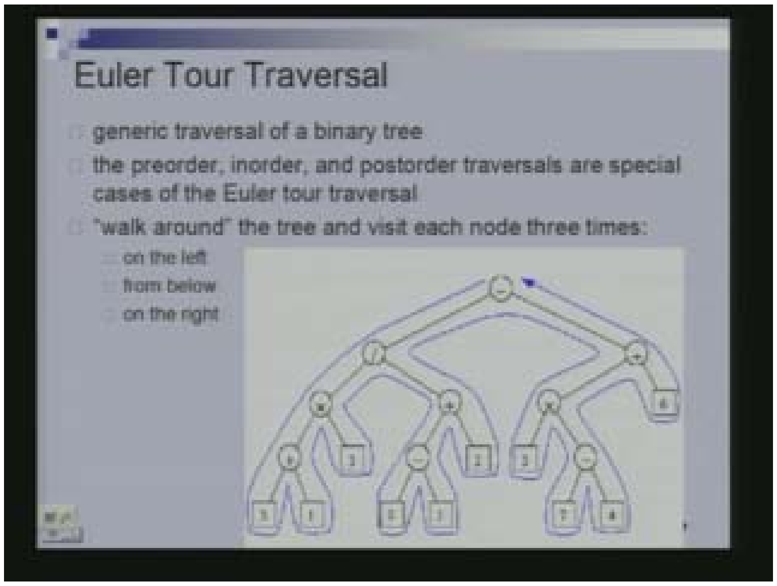

Inorder Example

- Inorder

  c b f d g a e

There is an another way of traversing a tree. That is called an Euler tour. Suppose this is a tree corresponding to an arithmetic expression. The tour is basically the one drawn in blue here. We start from here, keep going down. When we hit a leaf, we go up to his parent then go to the right sub tree and so on (Refer Slide Time: 21:54). It is a generic traversal of a binary tree. And all the 3 traversals that we have seen that is pre, post and inorder traversal can be viewed at as a special case of this Eulers tour traversal. Each node is basically getting visited thrice. Why because once we are coming like this (Refer Slide Time: 22:25) then other time we are going like this, touching this node. The third time we are coming like this and touching this node. So 3 times that we touch any node except the leaf nodes where you can count this as only once or thrice or what ever you want. But every internal node will be touched thrice. I should actually qualify, every internal node of degree which has 2 children, if the node has only one child then you will touch it only twice.

(Refer Slide Time: 21:31)



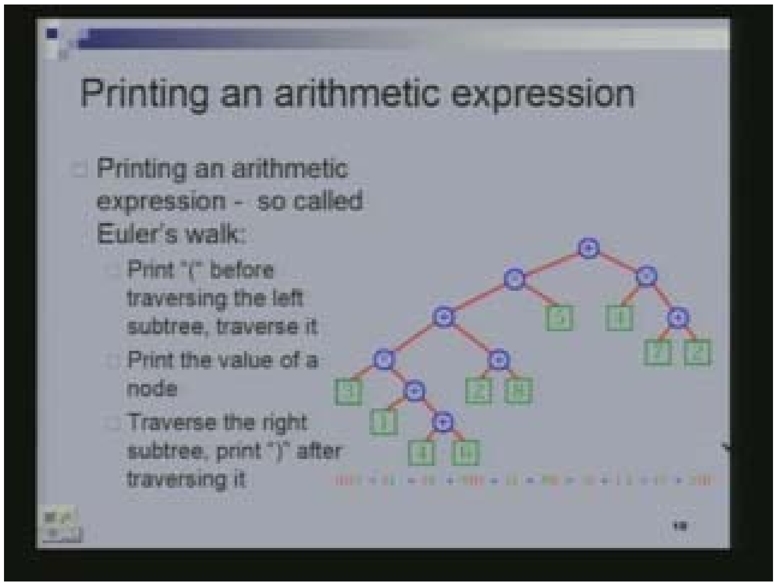
Euler Tour Traversal

- generic traversal of a binary tree
- the preorder, inorder, and postorder traversals are special cases of the Euler tour traversal
- "walk around" the tree and visit each node three times:
  - on the left
  - from below
  - on the right

Suppose this is the tree corresponding to a certain arithmetic expression which is given in the slide below. I want to print this arithmetic expression out with parenthesis. I want to draw the parenthesis, I want to print it out in this manner which is given in the below slide. So I can do an Euler walk to print this thing. Suppose I am here in the middle, before I start on the left sub tree I will print a left bracket. When I finish with the right sub tree I will print the right bracket. This right subtree corresponds to taking this path up (Refer Slide Time: 23:54), going up like this and when I am coming like this and touching this node will just print out the content of the node. Did you understand what I am saying? So recall that every node was visited thrice.

So once when I am visiting it from the left then essentially I am going to print a left bracket. When I am touching it from the right essentially I am going to print the right bracket and when I am touching it from below I am going to print the content of this node. If you do that then you will get exactly the one which is given in the bottom of the below slide. First I will touch the node at the center from the left so first I will print left bracket then I will touch this node on the left I will print another left bracket, I touch this node on the left I print another left bracket and I touch this node on the left so I print another left bracket. So I get 4 left brackets to begin with. Then I come to the node 3. For the leaf I will just print the content of the leaf and do nothing else so I just print 3. Then I am going to touch this node from below (Refer Slide Time: 24:50), I am just going to print a star or a multiplication. Then I come and touch it from the left so I print a left bracket then I come here which is 1 and so on. So you can think of this as essentially printing out this arithmetic expression as some kind of Euler walk on this tree.

(Refer Slide Time: 23:12)



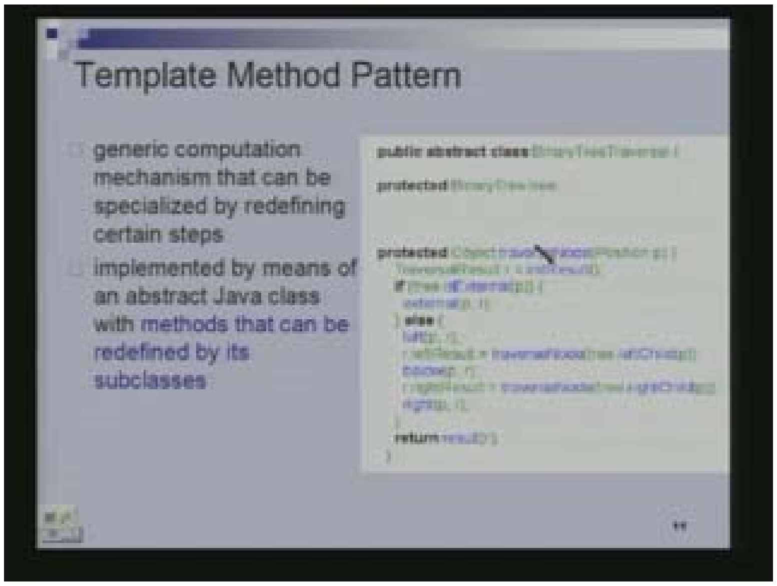
Printing an arithmetic expression

- Printing an arithmetic expression - so called Euler's walk:
  - Print "(" before traversing the left subtree, traverse it
  - Print the value of a node
  - Traverse the right subtree, print ")" after traversing it

We can actually write a generic method for tree traversal and then specialize it for whatever particular application you have. Whether you want to do a preorder, postorder or inorder traversal or any such thing. So this is just a small example. So you want to traverse a node specified at this position p. If this is an external node then you will call this method which is called external. You have not done anything here, we have just specified certain methods. External is a method that you will invoke if the node that you are trying to traverse is an external node. An external node is the same as a leaf node. So if it is a leaf node then that is the method you invoke. When you touch the node from the left then you will invoke this method called left. Here you continue with the left child.

When you touch the node from below, you will invoke this method then you continue with right child. Then when you touch the node from the right you will invoke this method. Now you can specify what these methods are. So by specifying these methods you can create the traversal of choice, you can specialize this binary tree traversal, this generic tree traversal. If you want to go into java details this is an abstract class which means that these methods in particular init result, isexternal of course is specified but init result, external, left, below, and right are left unspecified. In a class you leave certain methods unspecified then it becomes an abstract class because you cannot really create an object of that class anymore. But then at some point you can specify those methods. And in that manner create a sub class of this abstract class which specifies these methods. And in that manner specializes this generic tree traversal procedure. This is a generic tree traversal procedure. So if I were to specify left, below and right in a certain manner then I could get a class for printing out arithmetic expressions.

(Refer Slide Time: 25:16)



Template Method Pattern

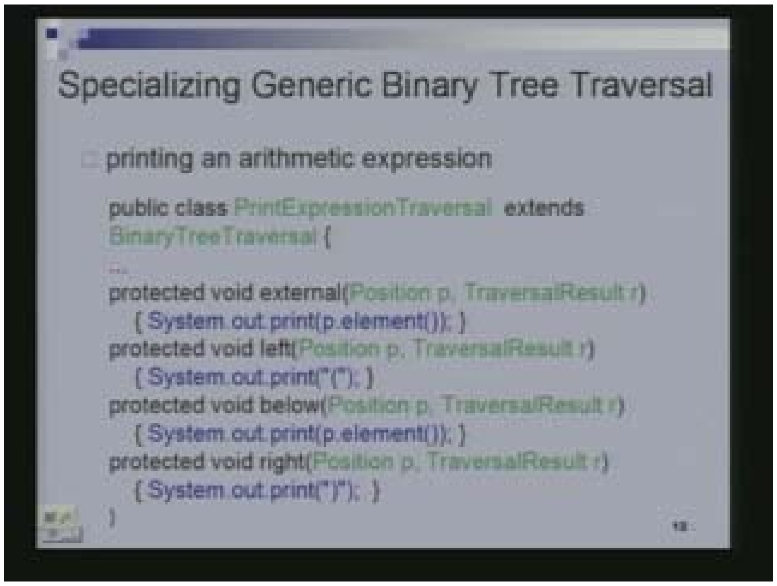
- generic computation mechanism that can be specialized by redefining certain steps
- implemented by means of an abstract Java class with methods that can be redefined by its subclasses

```
public abstract class BinaryTreeTraversal {

protected BinaryTree tree;


protected Object traverseNode(Position p) {
  TraversalResult r = initResult();
  if (tree.isExternal(p)) {
    external(p, r);
  } else {
    left(p, r);
    r.leftResult = traverseNode(tree.leftChild(p));
    below(p, r);
    r.rightResult = traverseNode(tree.rightChild(p));
    right(p, r);
  }
  return result(r);
}
```

11

You want to know what the left result was. When I come back from the left child, may be I compute a certain result. In that example of disk usage we wanted to compute what is the total space occupied by that directory. We computed the space required by the left child, the directory in the left child. Corresponding to the left child we computed the space required by the directory corresponding to right child. When we computed that, those could be stored in r.left result and r.right result and then we would compute their sum. That would be your final value, which would be the value we would return.

(Refer Slide Time: 28:48)



Specializing Generic Binary Tree Traversal

- printing an arithmetic expression

```
public class PrintExpressionTraversal extends
BinaryTreeTraversal {
...
protected void external(Position p, TraversalResult r)
   { System.out.print(p.element()); }
protected void left(Position p, TraversalResult r)
   { System.out.print("("); }
protected void below(Position p, TraversalResult r)
   { System.out.print(p.element()); }
protected void right(Position p, TraversalResult r)
   { System.out.print(")"); }
}
```

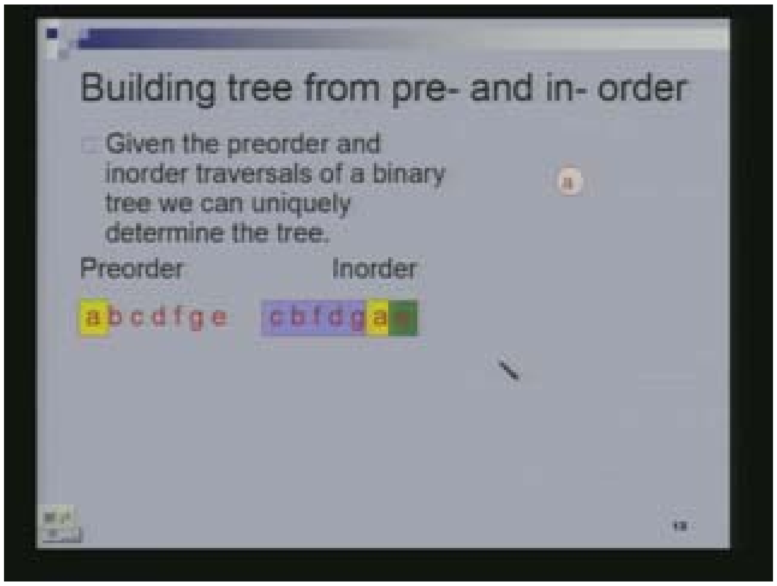
12

Let see how to specialize this for our printing arithmetic expression example. So recall that if the node is a leaf node then all we said was that we are going to print the content of that node. That is what we are saying, just print out the element in that node. P is a position just print out the element in that node, what ever the element may be. When we touch a node from the left, we said just print out a left bracket. So that is what we are saying just print out the left. When we touch a node from below, we just said that whatever is the operator present their just print that out. That is exactly what we are doing. When we touch it from the right, we just print out the right bracket.

The PrintExpression Traversal is the class which is extending BinaryTree Traversal. When I invoke the traversal method it will now print out the arithmetic expression with the tree corresponding to arithmetic expression in parenthesized form. So I could specialize the same class BinaryTree Traversal to use it to compute the total space occupied by certain directories structure, by specializing these methods in a slightly different manner.

Let us continue with our discussion on pre and in order. Suppose I give you the preorder and inorder traversal of a binary tree. I have mentioned this in the following slide. Can you use this to figure out what the tree is? Yes or no. Suppose this was the preorder traversal and this was the inorder traversal of the binary tree. I have given you these 2. Can you use these, to print out to tell me what the tree is? Yes you can. Why we can do that?

(Refer Slide Time: 32:33)



Building tree from pre- and in- order

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree.

Preorder Inorder

a b c d f g e c b f d g a e

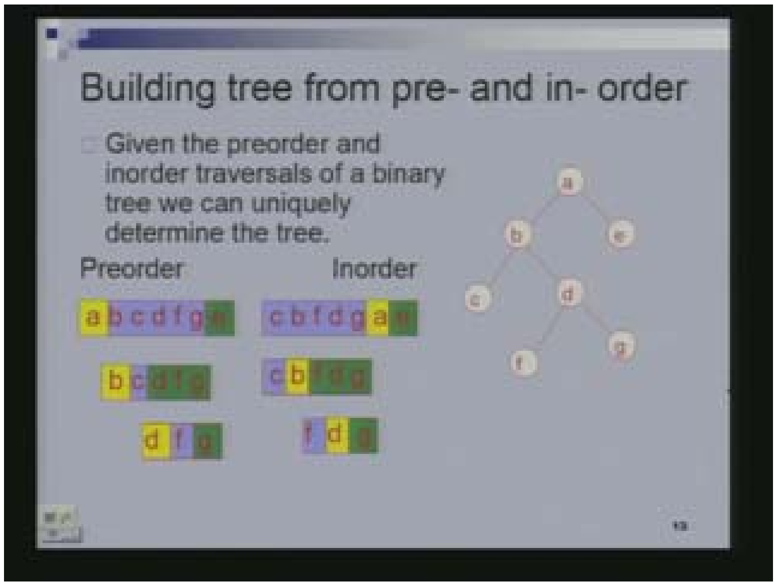
So given this preorder traversal, what can I say first? I have marked a as the root. What should I do? The b is left child. Is this true that b is the left child? No, not necessarily. The root might not have a left child at all. That can happen, so we cannot say anything. All we can say is, a is the root. Let us find a in the inorder traversal. The a is the root, I know that let me just put down the node for the root. And I know that I will search for a in the inorder traversal. Now what do I know? I know that e is to the right that is e is the right sub tree. I know that this (e) is the inorder traversal of the right sub tree and this blue colored is the inorder traversal of the left sub tree of a. The green is the inorder traversal of the right sub tree and blue is the inorder traversal of the left sub tree.

I know that the left sub tree has 5 elements in it. This is the information I know. So in the preorder traversal the 5 elements following a, would correspond to the preorder traversal of the left sub tree. The one element following that would be the preorder traversal of right sub tree. So in essence what have I managed to do? I have managed to identify what the left and right sub tree are and I know their preorder and inorder traversals. I know the preorder traversal of the left sub tree and I know the inorder traversal of the left sub tree. So my problem, I can use recursion now. I know e is the right sub tree and now I can basically work on this problem, where I am given the preorder and in order traversal of a tree and I need to figure out what the tree is. And whatever is the tree is, I will come and plug it as the left sub tree of a.

Let us continue with the example and let see what we will do now. So b is the root of this left sub tree. Whatever argument we used before. So b is the root of the left sub tree, we are going to see where b is in the inorder traversal. The b is here and b is the root. So this (c) would be the left sub tree of b now. And f, d, g is going to be the right sub tree of b. So on the left I have only one element that would be c here. So c would be the left sub tree and this (d, g, f) would be the preorder traversal of the right sub tree. I figured out the c, since on the left there is only one so I can put that c down. I do not know what the right subtree is. I just know preorder traversal and inorder traversal. I know that it has 3 nodes, the right sub tree has 3 nodes. So the problem recursively reduces to this problem of given the preorder and inorder traversal of this 3 node tree, I need to figure out what the trees is.

So once again I know that the root is going to be d. I look for d in the inorder traversal, it is there. I know on the left I have f and on the right I have g. I know the root is d and I know that the left sub tree would have f and the right would have g. I get something like the one which is given in the below slide.

(Refer Slide Time: 30:36)

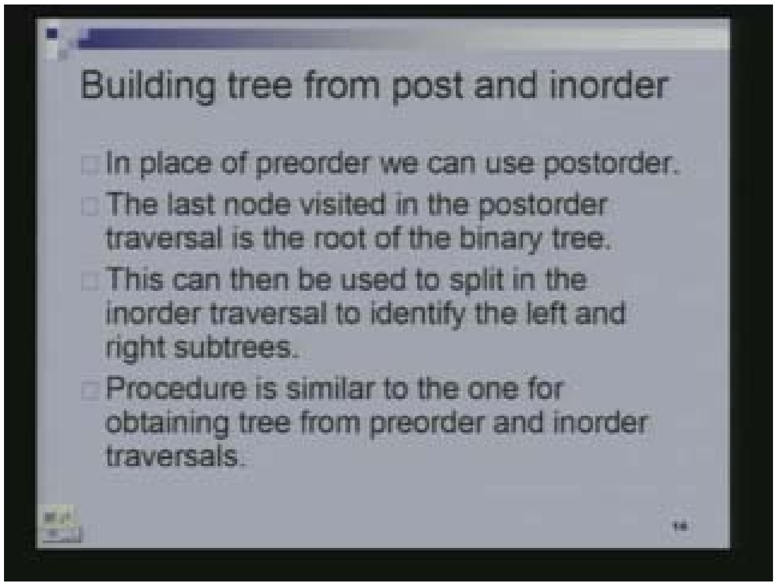

Building tree from pre- and in- order

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree.

You can translate this in to a piece of code. It will require some thought because you have to write it in a recursive manner. This is your second assignment which I have put it upon the web today. This would be some input, you will take from the user, the pre order and the inorder traversal. And you have to compute not the tree but you have to compute the post order traversal which is simple if you have computed the tree. Because you can do a post order traversal of this tree now and give the result. It might be possible that given any arbitrary sequences, it is not necessary there is a binary tree corresponding to that. You will also have to flag out an error, if the sequences that given to you are such that they could not possibly be the pre and inorder traversal of any binary tree. This is pre- and in- order.

Given post and in also you should be able to compute the tree. Suppose I give you the postorder, inorder. Can you use that to compute the tree? Recall what did we do? We try to first figure out where the root was. In the preorder the very first element is the root, in a postorder the last element is the root. So first, once you know the root then you will search for the root in your inorder traversal and wherever you find that root, that neatly divides the thing into a left sub tree and right sub tree. Once you know the number of nodes in the left sub tree, you can figure out what the post order traversal of the left sub tree. If there were 5 nodes in the left sub tree then the first 5 nodes of your post order traversal would be the post order traversal of the left sub tree. So in this manner again you can recursively work. You can recursively figure out what the left and right sub trees are and then plug them up to the tree. Given a post and in order traversal also you can do. What is the third question? Given pre and post can you do it? No.

(Refer Slide Time: 36:15)



Building tree from post and inorder

- In place of preorder we can use postorder.
- The last node visited in the postorder traversal is the root of the binary tree.
- This can then be used to split in the inorder traversal to identify the left and right subtrees.
- Procedure is similar to the one for obtaining tree from preorder and inorder traversals.

14

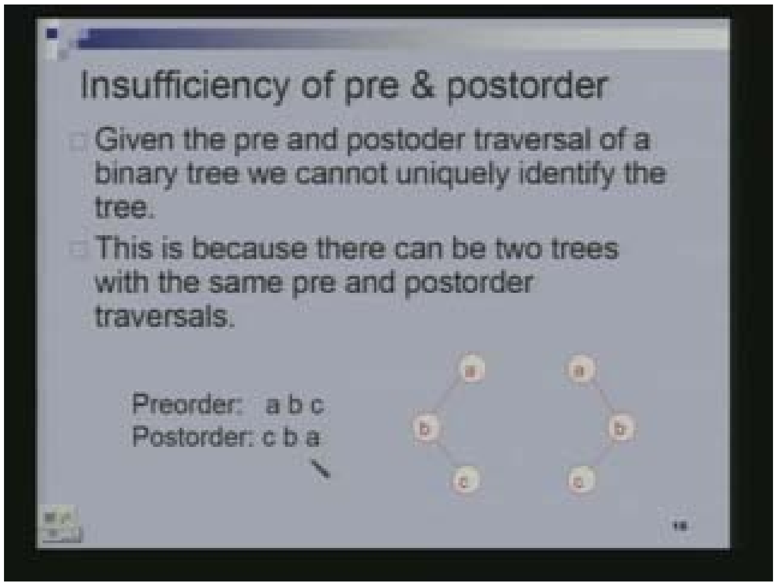
Given the pre and post order traversal of a binary tree you cannot uniquely determine the tree. And the reason for that is there can be 2 trees which have the same pre and post order traversal.

Preorder: a b c

Postorder: c b a

Suppose I gave you the above preorder traversal and postorder traversal that is a 3 node tree. What do you think is the tree? The tree given in the below slide is the tree with preorder traversal a b c. The postorder traversal will be c b a. This tree also has the same pre and post order traversal. The a b c is the preorder traversal, the postorder traversal is c b a. Which of this tree is a right one? Both of them. There is no unique tree, we could also have c as the right child. So these are only 2 but they could be many trees that you could construct.

(Refer Slide Time: 37:44)



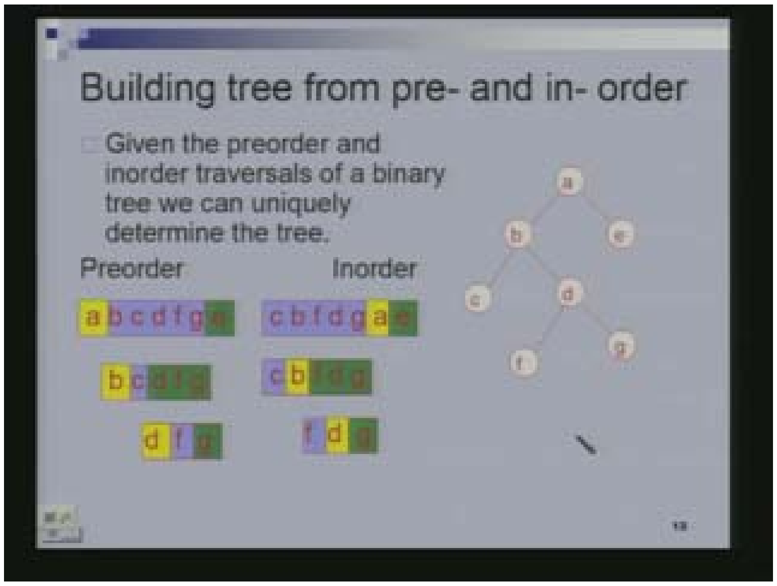
Insufficiency of pre & postorder

- Given the pre and postoder traversal of a binary tree we cannot uniquely identify the tree.
- This is because there can be two trees with the same pre and postorder traversals.

Preorder: a b c
Postorder: c b a

That is the problem. Given a pre and post order traversal you cannot uniquely determine the tree. Because there could be many different trees with the same pre and post order traversals. But here note that some nodes in the trees had only one child. Suppose I gave you this information that every internal node of the tree has 2 children, not complete but every internal node of the tree has 2 children. What is that called? The below given slide is an example of a tree in which every internal node has 2 children. This is not a full tree, this is just a tree with every internal node having 2 children. There is no name to it.

(Refer Slide Time: 39:32)



Building tree from pre- and in- order

- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree.

Preorder Inorder

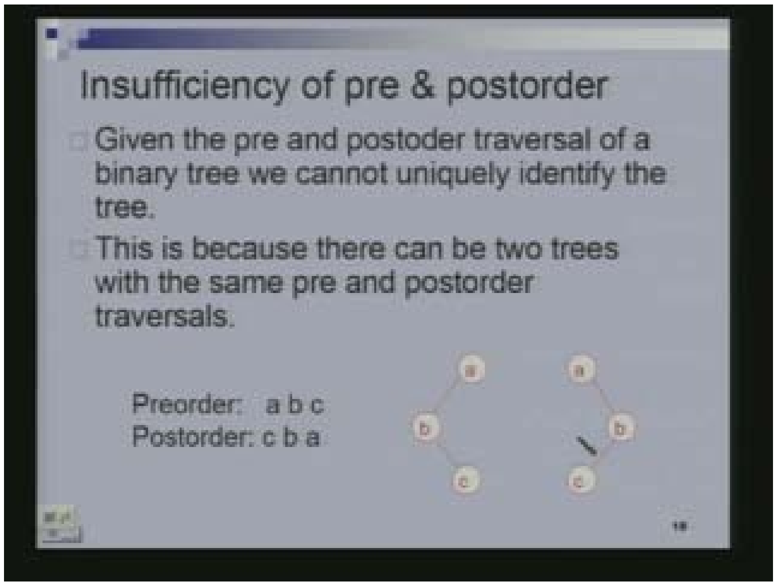
a b c d f g e   c b f d g a e

b c d f g   c b f d g

d f g   f d g

Suppose I gave you this information. Did you say an Indian tree? If each internal node of the binary tree has exactly 2 children. Then actually you can use the pre order and post order to determine the tree uniquely again. Let see just as an exercise, why this is true? I gave you the pre and post order traversal. What is the first thing you can say? The a is the root. We can quickly draw the root. What can you now? The b has to be the left child.

(Refer Slide Time: 39:51)

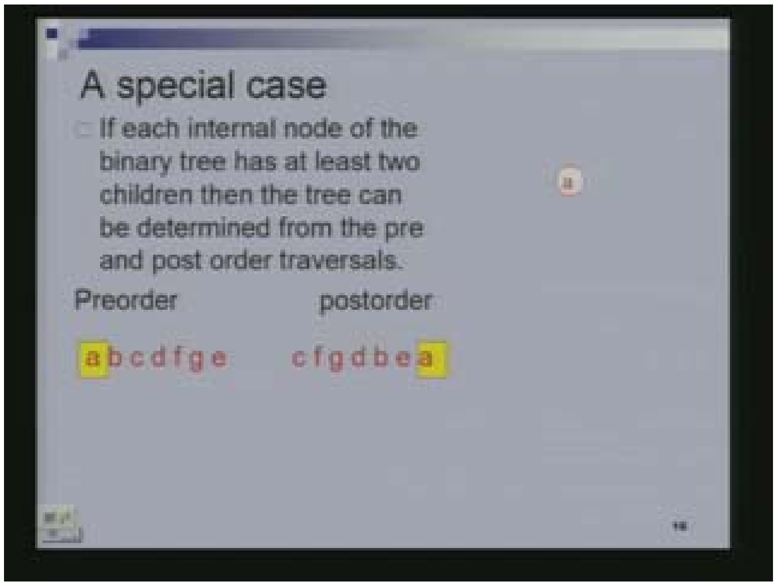


Insufficiency of pre & postorder

- Given the pre and postoder traversal of a binary tree we cannot uniquely identify the tree.
- This is because there can be two trees with the same pre and postorder traversals.

Preorder: a b c
Postorder: c b a

(Refer Slide Time: 40:04)



A special case

- If each internal node of the binary tree has at least two children then the tree can be determined from the pre and post order traversals.

Preorder postorder

a b c d f g e c f g d b e a

It has 2 children. So we know that there is something on the left or actually I have done the example on the right. What can I say is the right child of a? The a has right child because every node has 2 children. What is the right child of a? It has to be e. Since my example, I have worked it out with that. The e, I know is the right child from the picture given in the above slide. I see e in the above picture. And after e nothing is there. What that it say? That says e is a leaf because in a preorder traversal I would have first visited e and then gone to its children. But there is nothing following it which means e has no descendants.

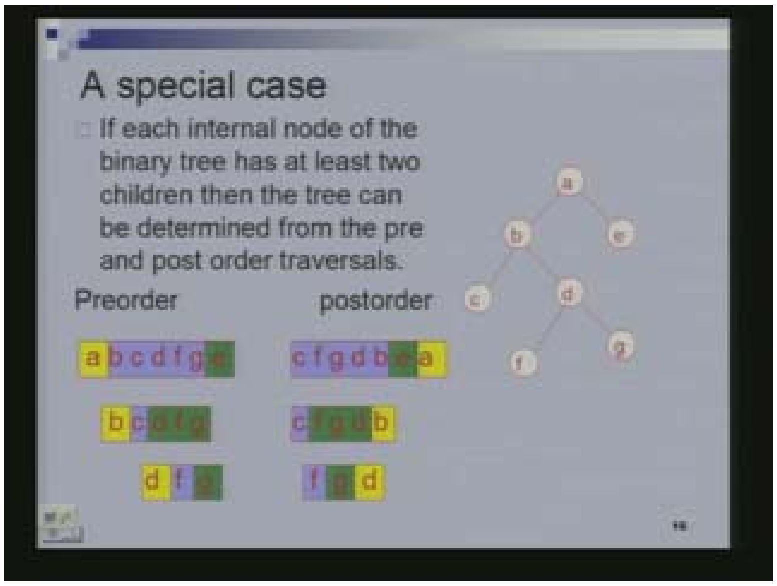
Basically it means that the left sub tree has b c d f g. So e is the right child and the left sub tree has b c d f g as the preorder traversal and c f g d b as its postorder traversal. The same thing, since I have managed to divide, I am going to continue. So I have b c d f g as the preorder traversal of the left subtree and c f g d b as the postorder traversal of the left subtree. Once again I know that the root is b, I have drawn the root. What do I know about the right child of b? I know the right child is d. I see where d is in the case of preorder traversal. So d is the right child, every thing that follows d has to be in the right sub tree. Everything that follows the d in the pre order traversal has to be in the right sub tree. Why because in the right sub tree when I did a pre order traversal I have first visited the d and only then visited the other elements. So d f g forms the right sub tree and c forms the left sub tree. The left sub tree has only one element, that is what I am going to do here in the case of postorder traversal.

This one element c would be my left sub tree and this f g d would be my right sub tree. I have figured out the left sub tree which has only single element, I can just draw it. And now I know that right sub tree has nodes d f g where preorder traversal is d f g and postorder is f g d. I know root is d, actually I have already drawn that out. I know the right child is g. The g is here and there is nothing beyond that means g has no children further. I can draw that out and then I know what remains.

The left sub tree has only one node, it is in the left. You can do the same thing, I am just showing it to you at high level. In fact this is the level I will be following for all the algorithms we do in this class. You will have to translate it in to code. What you have to do is recursion or whatever it is. The idea of the assignment would be that. Many of these algorithms you should learn or you should figure out how to best program efficiently. You can write down the code also for this given the pre order and post order traversal of a binary tree every internal node of which has exactly 2 children, you can even use that to figure out what the tree is.

Why did we work with 2 traversals? Why can not we just take pre order? So given just a pre order traversal can I use that to figure out what the tree is? This has the pre order traversal. The a, b, c, d, f, g, e was the pre order traversal of this tree. This is one tree with this as a pre order traversal. Can you think of another tree which has this as the pre order traversal? I could just have all of the nodes in one line to say, a b c and d below it, f after it g after at it, e at the very end.

(Refer Slide Time: 41:04)



A special case

- If each internal node of the binary tree has at least two children then the tree can be determined from the pre and post order traversals.

Preorder postorder

a b c d f g e c f g d b e a

b c d f g c f g d b

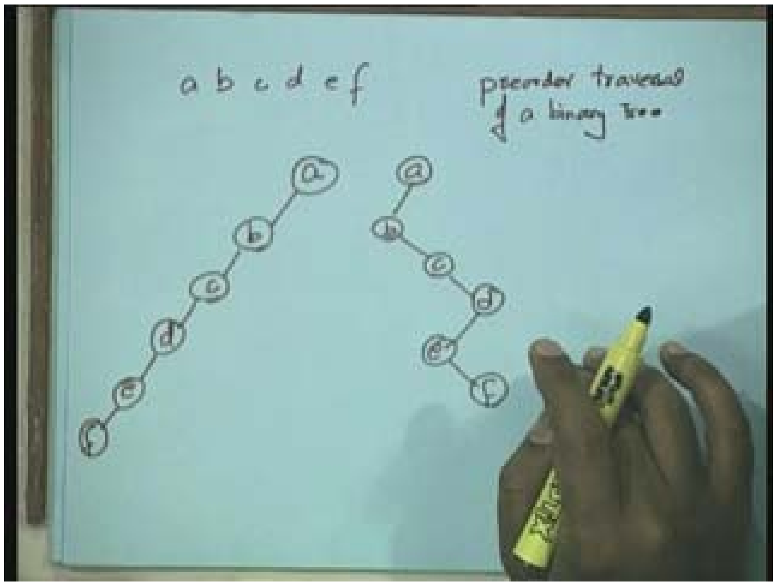
d f g f g d

I could get huge number of different combinations all of which would give rise to the same preorder traversal. So with just the knowledge of one traversal you cannot really do much. Similarly for post order similarly for in order, even if I were to give you the in order traversal of this you can construct many different trees with the same in order traversal. So just using one, you cannot really do anything but 2 suffices for most purposes. In the only case when 2 does not suffices when you are given the pre order and the post order traversal. If some internal nodes of your binary tree could have only one child. But if you were told that every internal node has 2 children then even that is sufficient.

Can you count the number of binary trees given only lets say a preorder traversal? What do you think? At least 2 to the power n-1. We said a, b, c, d, e and f is my pre order traversal of a binary tree.

We said a, b, c, d, e and f. This is one and all you are saying is I can make each one of them either a left or a right child of its parent. So this could be one the other option could be at the side a, b, c, d, e, and f and so on. So just since each of the nodes can be either the left or right child of its parent. But there are many other possibilities. This is absolute minimum. You can have many other possibilities. There could be lots and lots of things possible. Of course it will be finite because there are only finitely many different trees with 6 nodes on them. It will be of some finite number, it will be just a function of n exactly. But I do not know how to compute the close form expression for such things.

(Refer Slide Time: 49:38)



So with that we will stop today’s class. What we looked at was tree traversals. How to traverse? So 3 different ways of traversing trees, in order, pre order and post order traversal for binary trees and for general trees there is no notion of an in order traversal as you perhaps understand. Why there is no notion of an in order traversal for general trees? Because if a node has 3 children then when do you visit the node itself? After visiting the first child or after visiting the second child or when? But in a binary tree there is a notion of left and a right. First you visit the left then you visit the node, then you visit the right. So there is also a notion of an in order traversal. And we saw some applications of these. And how given inorder, preorder and postorder traversal, two of these traversals you can figure out what the tree was which gave rise to those traversals.