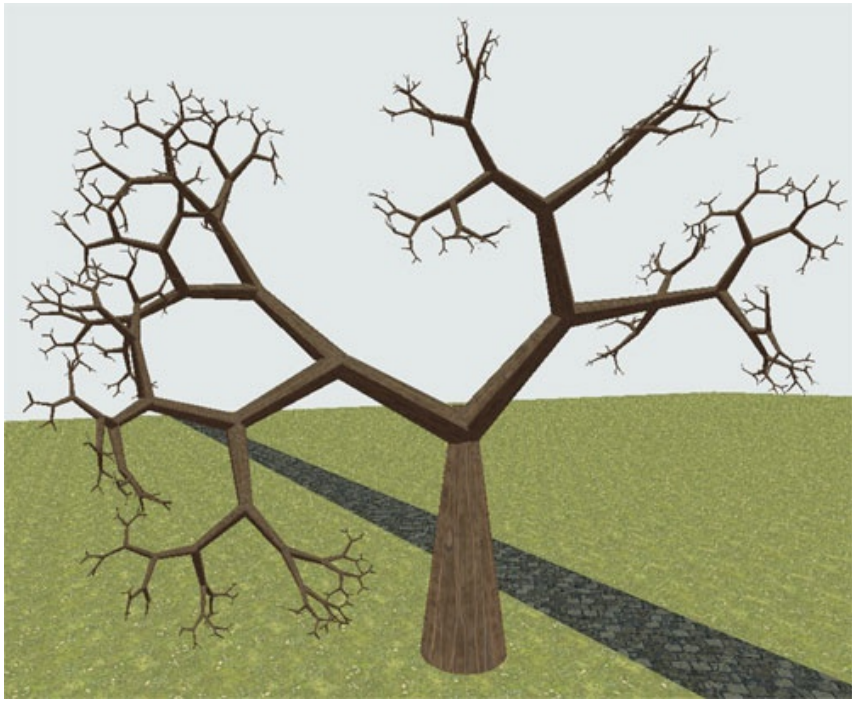
Tree Data Structure

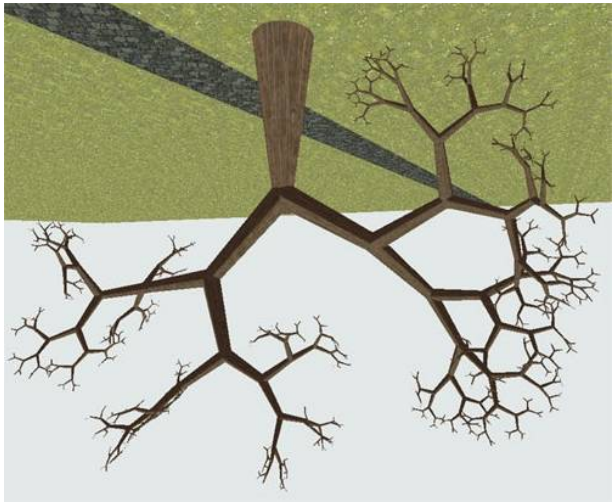
A.R. Routray
Fakir Mohan University

Trees

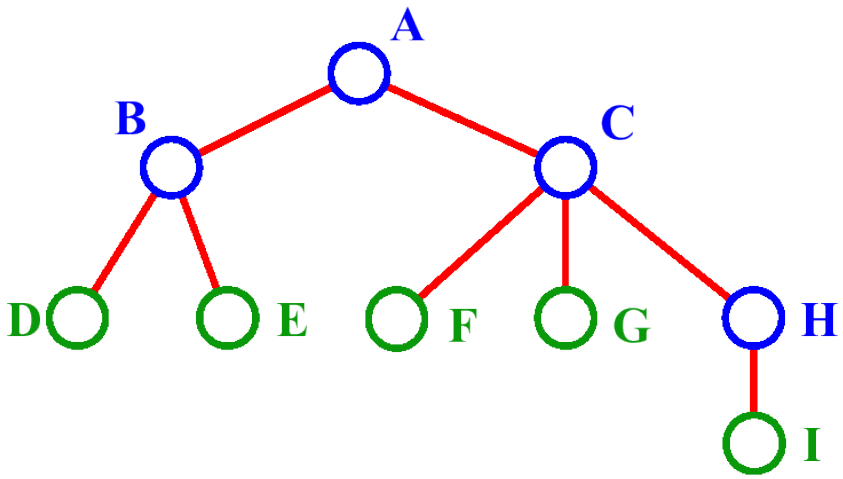
- Trees: Tree terminology
- Binary tree
- Tree traversals



Trees: Definitions

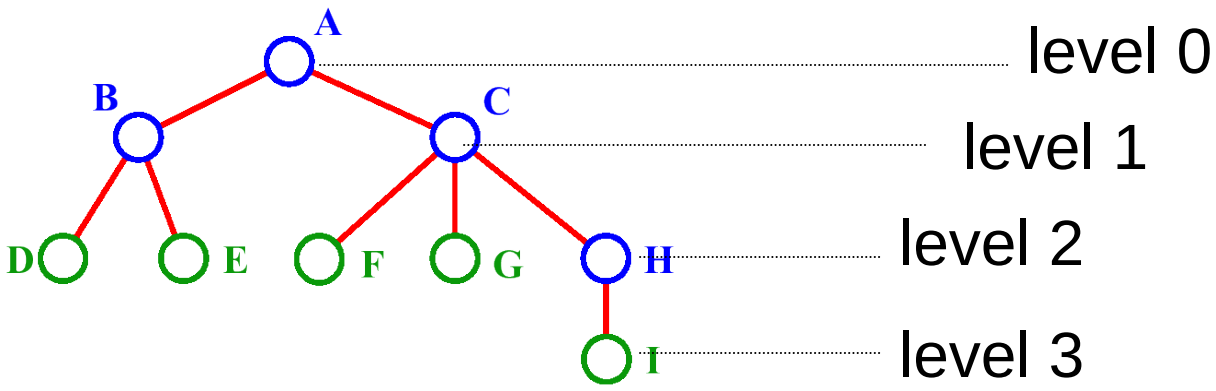


- *A* is the *root* node.
- *B* is *parent* of D & E.
- *A* is *ancestor* of D & E.
- D and E are *descendants* of *A*.
- *C* is the *sibling* of B
- *D* and *E* are the *children* of B.
- *D, E, F, G, I* are *leaves*.



Trees: Definitions (2)

- *A, B, C, H* are *internal nodes*
- The *depth (level)* of *E* is *2*
- The *height* of the tree is *3*
- The *degree* of node *B* is *2*



The number of edges from the node to the deepest leaf is called height of the tree.

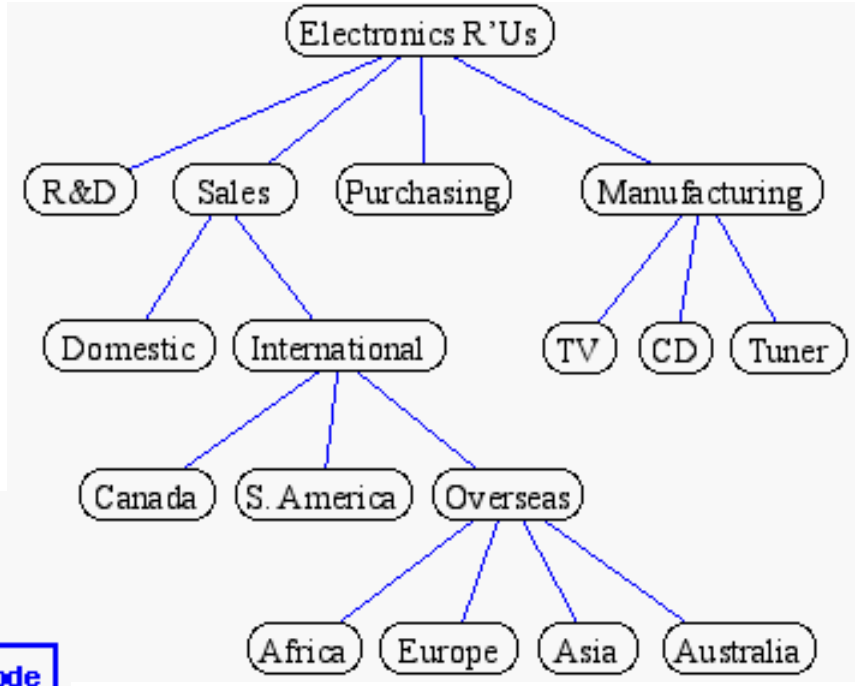
The number of edges from the root to the node is called depth of the tree.

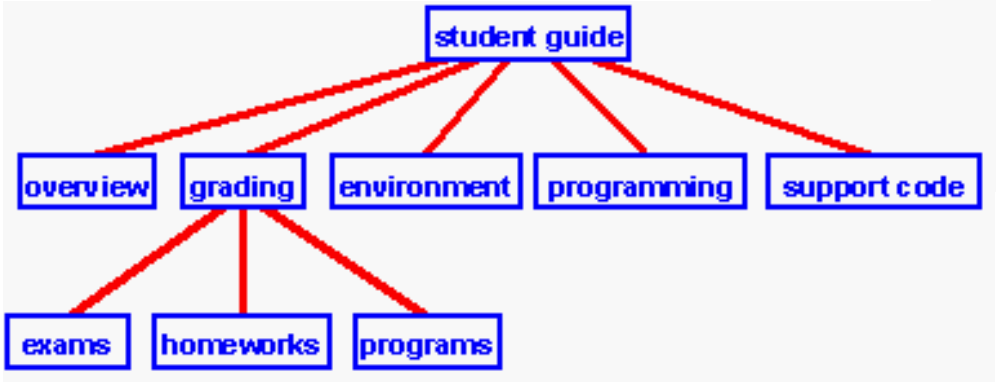
A full binary tree is a tree in which each node has exactly 0 or 2 children.

A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right is called complete binary tree. A Tree in which each node has exactly zero or two children is called full binary tree. A Tree in which the degree of each node is 2 except leaf nodes is called perfect binary tree.

Trees

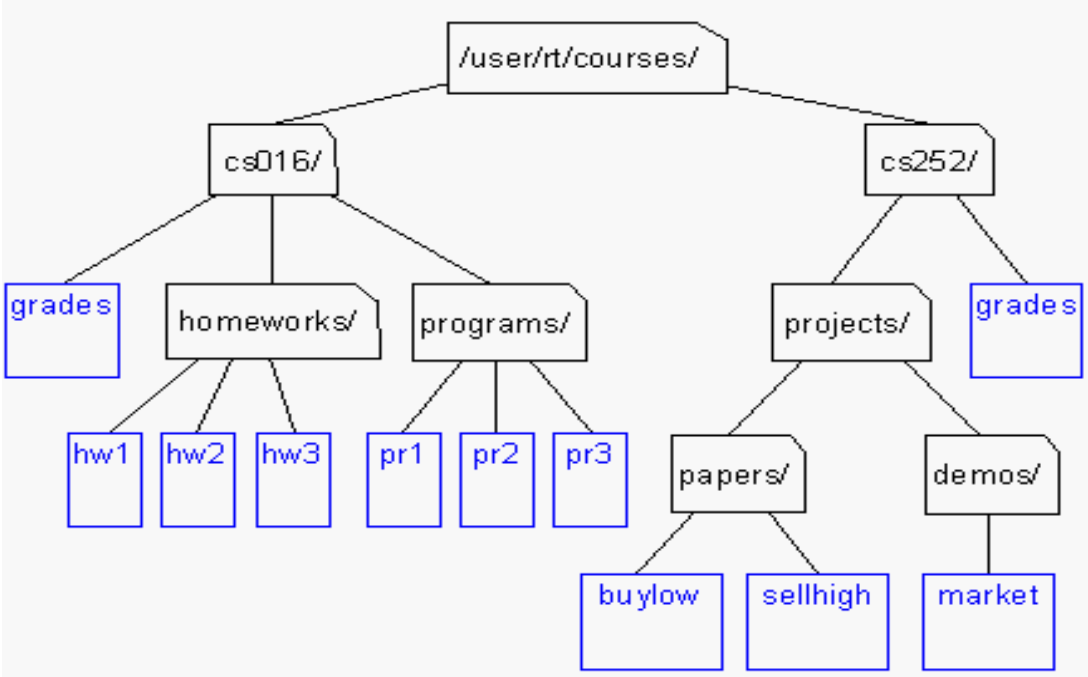
- A tree represents a hierarchy, for e.g. the organization structure of a corporation
- Or table of contents of a book





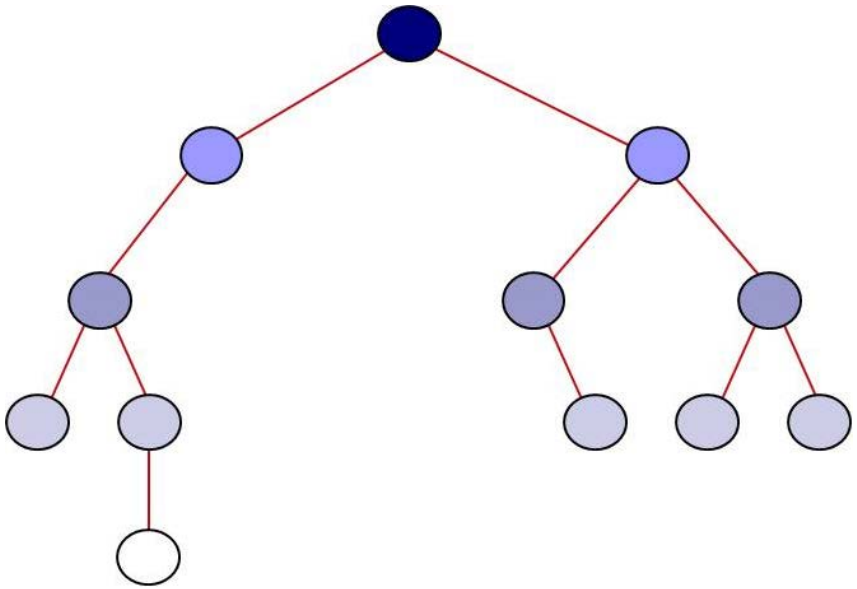
Another Example

Unix or DOS/Windows file system



Binary Tree

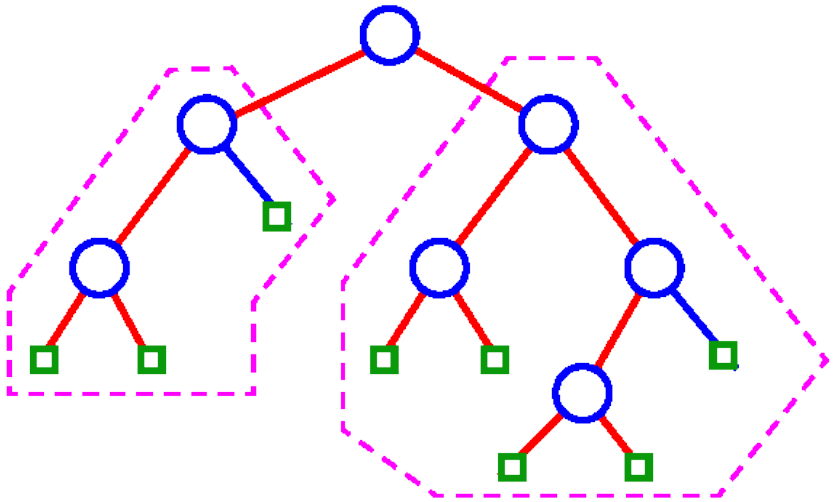
- An **ordered tree** is one in which the children of each node are ordered.
- **Binary tree:** ordered tree with all nodes having at most 2 children.



Recursive definition of binary tree

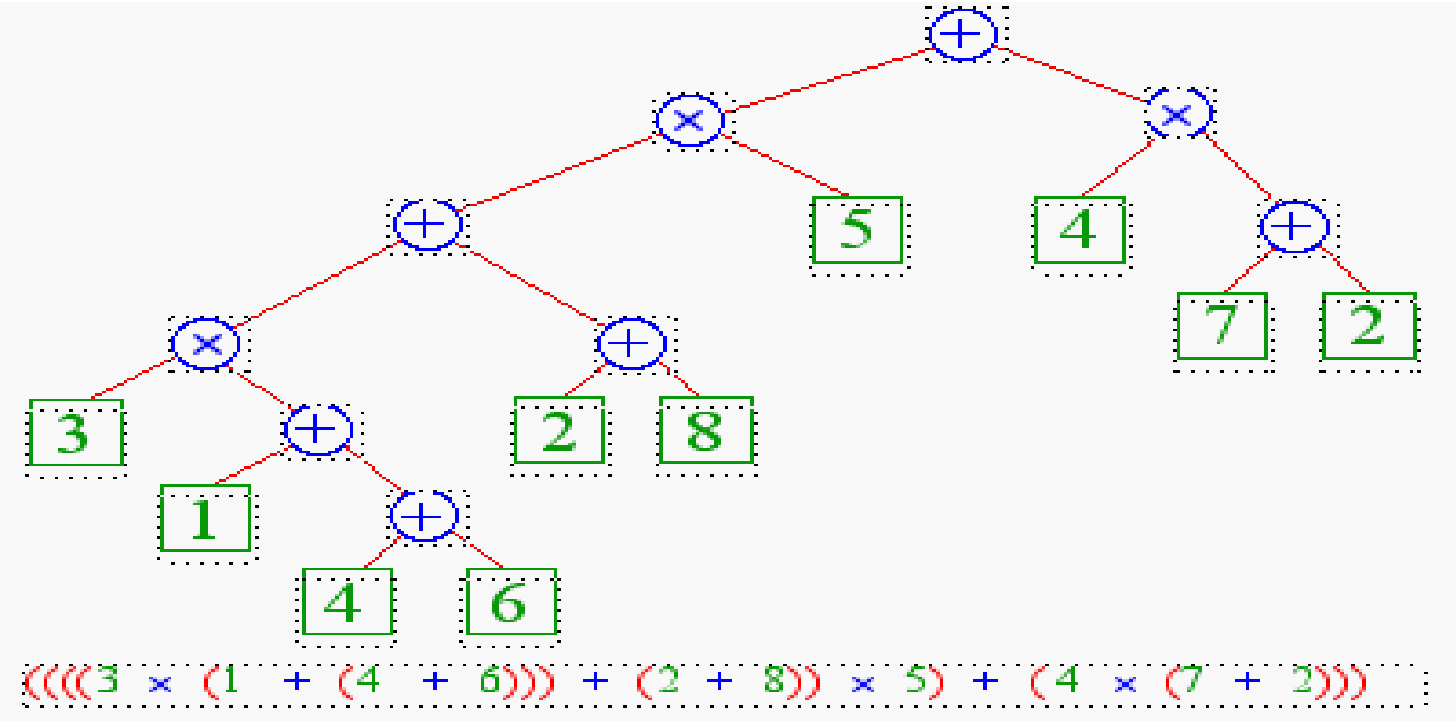
A binary tree is either a

- leaf or
- An internal node (the root) and one/two binary trees (left subtree and/or right subtree).



Examples of Binary Trees

arithmetic expressions

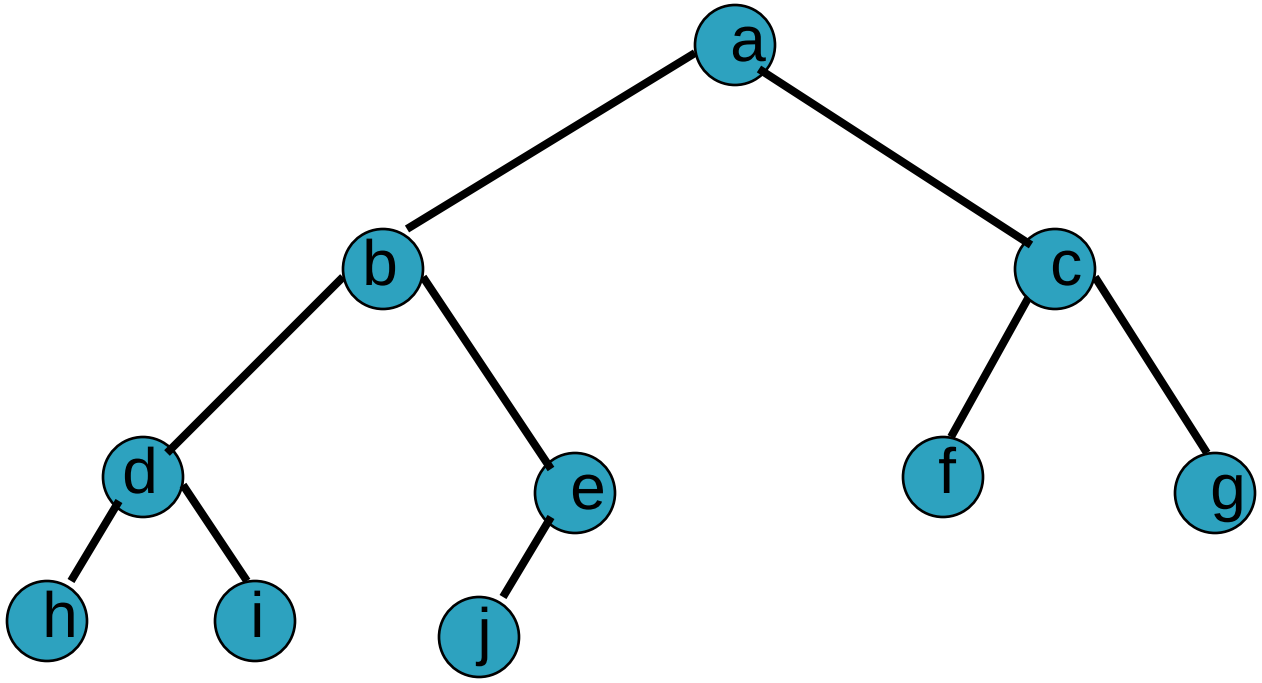


Binary Tree Representation

- Array representation.
- Linked representation.

Array Representation

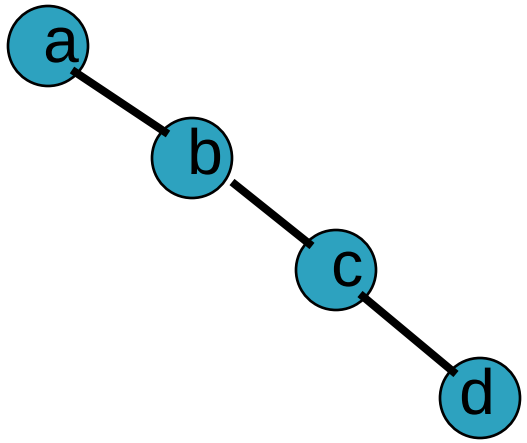
- Number the nodes using the numbering scheme for a full binary tree. The node that is numbered i is stored in tree[i].



tree[]

a	b	c	d	e	f	g	h	i	j

Right-Skewed Binary Tree



tree[]

a	-	b	-	-	-	c	-	-	-	-	-	-	-	d

- An n node binary tree needs an array whose length is between n+1 and 2^n.

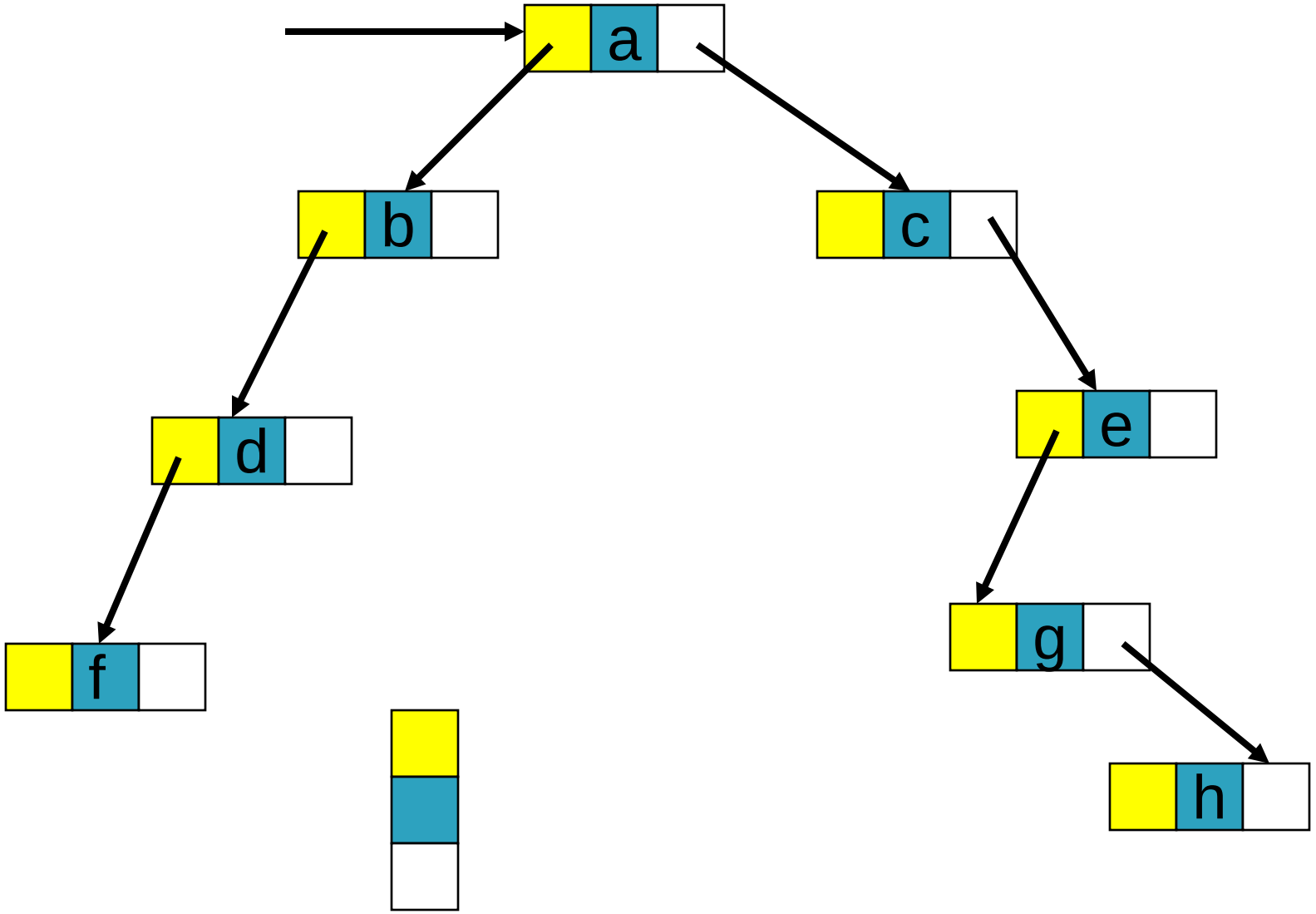
Linked Representation

- Each binary tree node is represented as an object whose data type is BinaryTreeNode.
- The space required by an n node binary tree is n * (space required by one node).

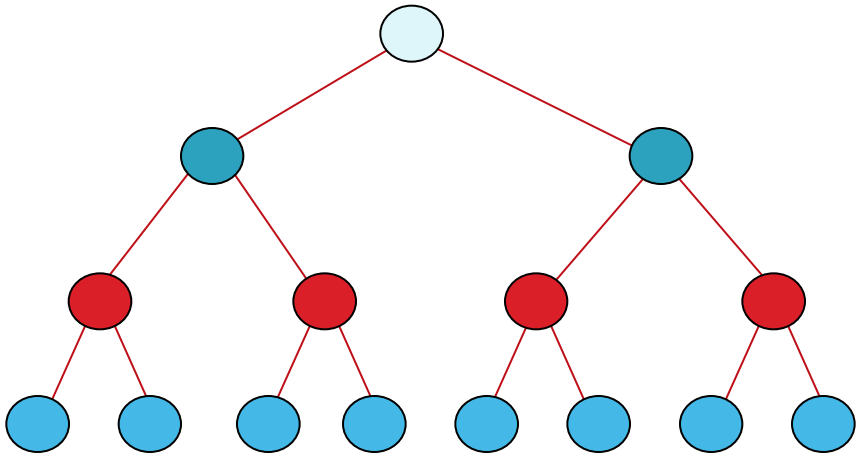
BinaryTreeNode

```
static class Node {
        int value;
        Node left, right;
        Node(int value){
            this.value = value;
            left = null;
            right = null;
        }
    }
```

Linked Representation Example



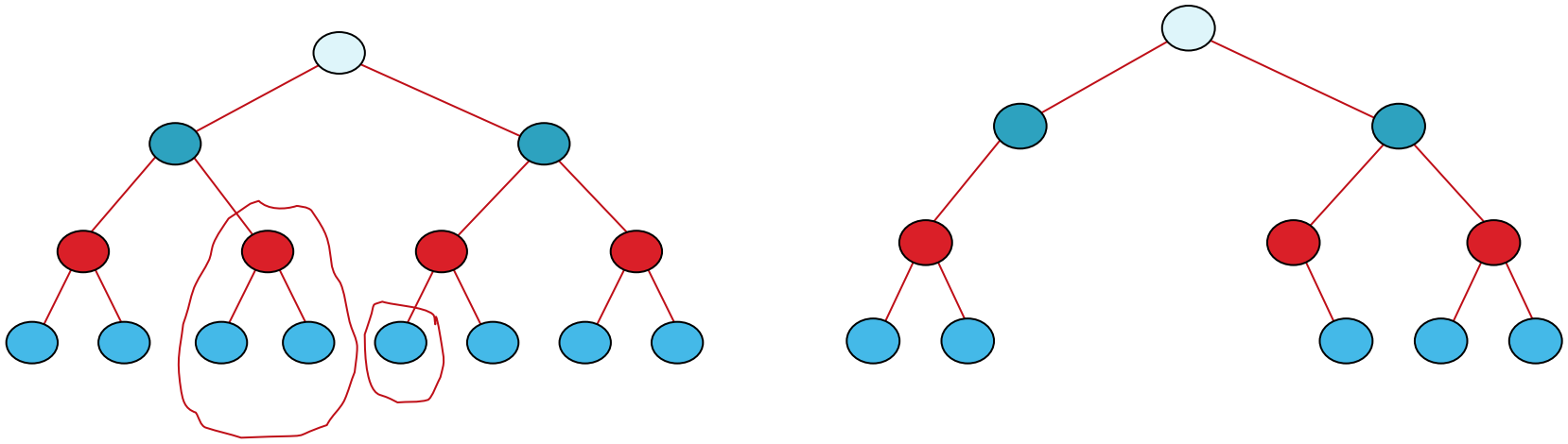
Complete Binary tree



- 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 = $1+2+2^2+\ldots+2^{h-1} = 2^h-1$
  - 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)

Binary Tree

- A Binary tree can be obtained from an appropriate complete binary tree by pruning

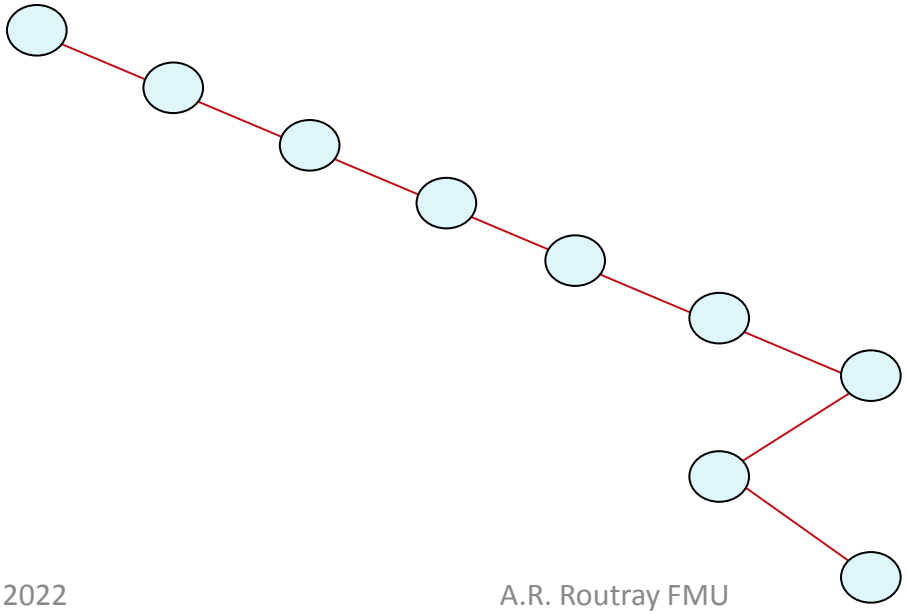


Minimum height of a binary tree

- A binary tree of height h has
  - At most 2^i nodes at level i
  - At most $1+2+2^2+\ldots+2^h = 2^{h+1}-1$ nodes
- If the tree has n nodes then
  - $n <= 2^{h+1}-1$
  - Hence $h >= \log_2 (n+1)/2$

Maximum height of a binary tree

- A binary tree on n nodes has height at most n-1
- This is obtained when every node (except the leaf) has exactly one child



No of leaves in a binary tree

- no of leaves <= 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 <= (k_1+1)+($k-k_1$) = k+1

leaves in a binary tree (2)

For a binary tree on n nodes

- No of leaves + no of internal nodes = n
- No of leaves <= no of internal nodes + 1
- Hence, no of leaves <= (n+1)/2
- Minimum no of leaves is 1

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

Traversing Trees (preorder)

- preorder traversal

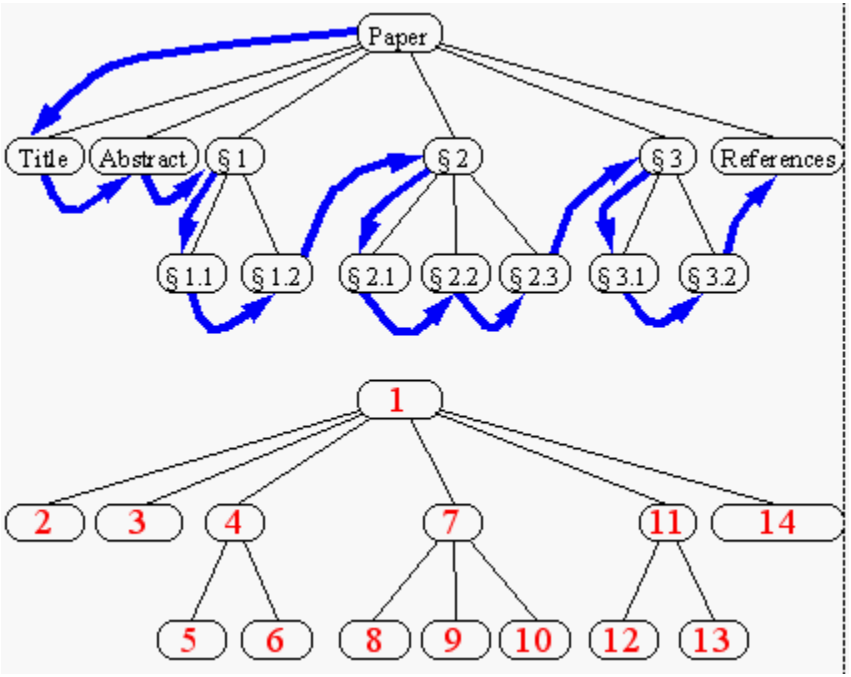
  Algorithm preOrder(v)

  “visit” node v

  for each child w of v do

  recursively perform preOrder(w)

- reading a document from beginning to end



Traversing Trees (postorder)

- postorder traversal

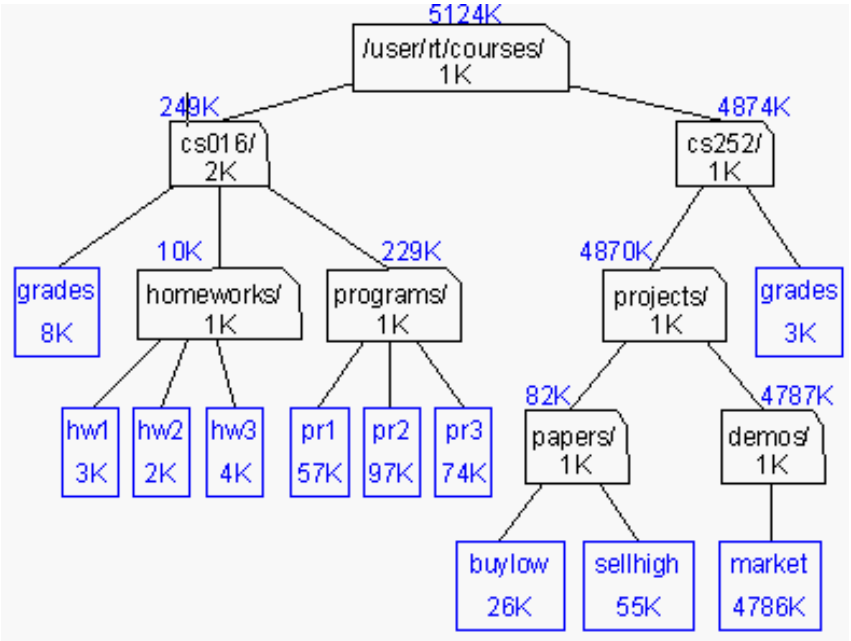
  Algorithm postOrder(v)

  for each child w of v do

  recursively perform postOrder(w)

  "visit" node v

- du (disk usage) command in Unix



Traversals of Binary Trees

```
public void inorder(Tree root) {
inorder(root.left);
System.out.println(root.data);
inorder(root.right);
}
```

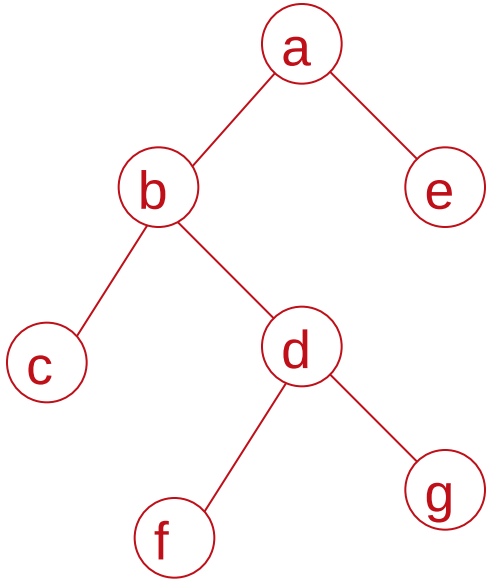
```
public void postorder(Tree root) {
inorder(root.left);
inorder(root.right);
System.out.println(root.data);
}
```

```
public void preorder(Tree root) {
System.out.println(root.data);
inorder(root.left);
inorder(root.right);
}
```

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

Traversing Trees

- Besides preorder and postorder, a third possibility arises when v is visited between the visit to the left ad right subtree.

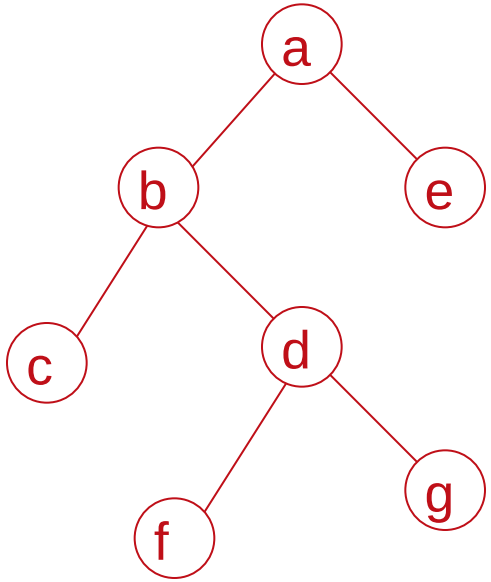
- **Algorithm** inOrder(v)

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

Inorder Example

- Inorder

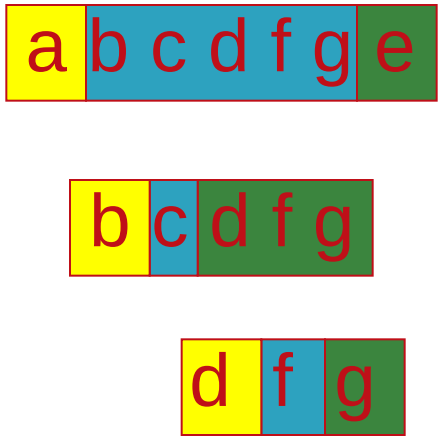
c b f d g a e



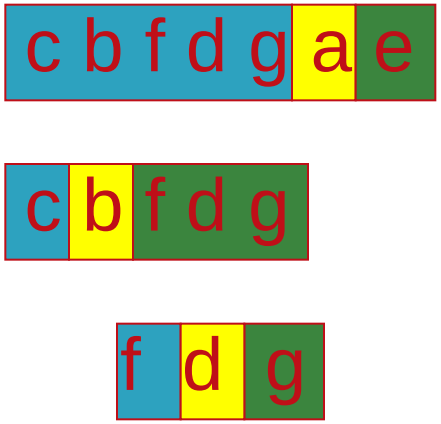
Building tree from pre- and in-order

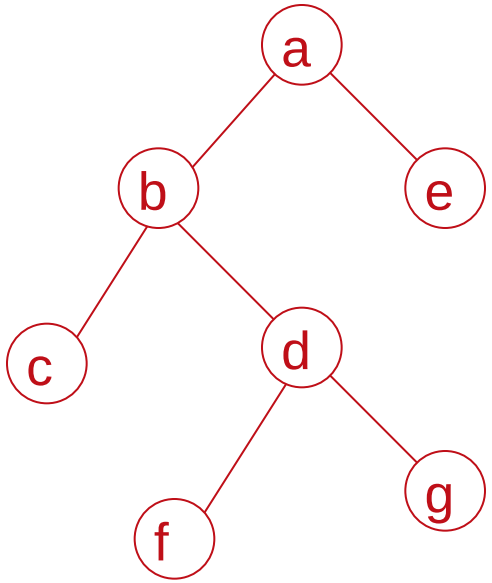
- Given the preorder and inorder traversals of a binary tree we can uniquely determine the tree.

Preorder



Inorder



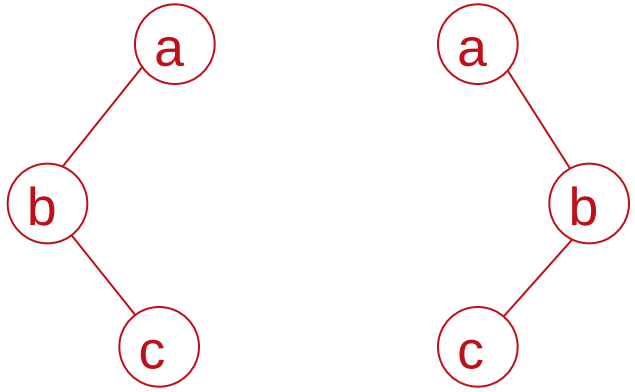


Building tree from post and inorder

- 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.

Insufficiency of pre & postorder

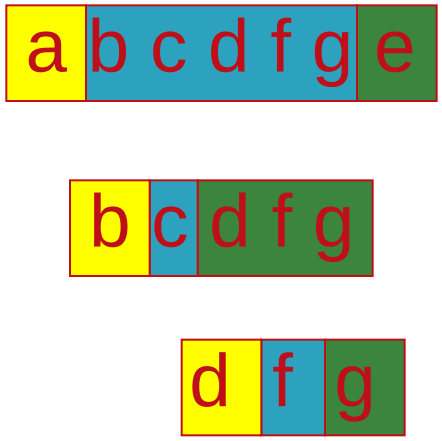
- Given the pre and postoder traversal of a binary tree we cannot uniquely identify the tree.
- This is because there can be two trees with the same pre and postorder traversals.
- Preorder: a b c
- Postorder: c b a



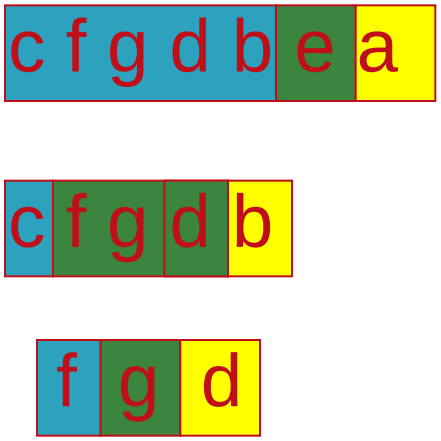
A special case

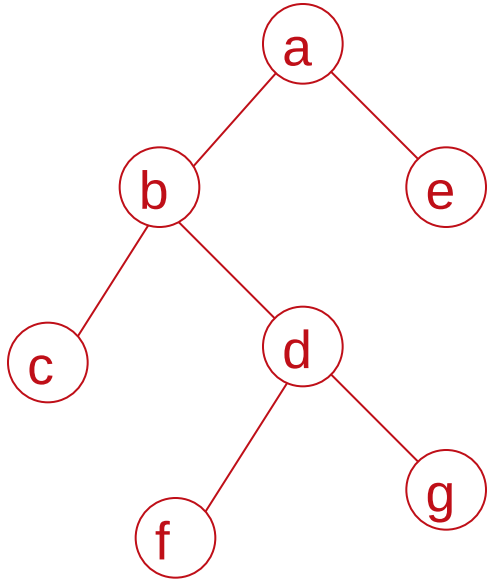
- If each internal node of the binary tree has at least two children then the tree can be determined from the pre and post order traversals.

Preorder



postorder





Example 1

- Inorder D,B,H,E,A,I,F,J,C,G
- Preorder A,B,D,E,H,C,F,I,J,G

➢ Draw the binary tree

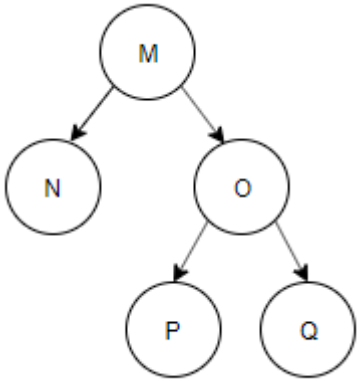
➢ Give a linear array representaion

➢ Node structure in C

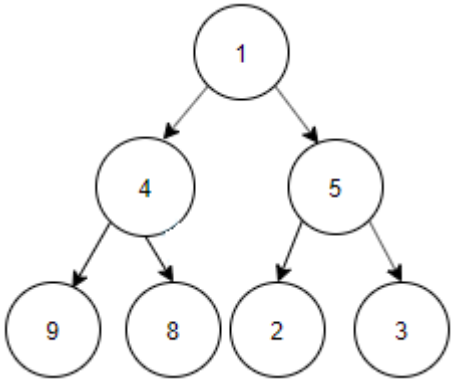
Example 2

- Given a complete binary tree with 7 nodes. The inorder traversal is GDEABCF. Find preorder and postorder traversals of the tree.

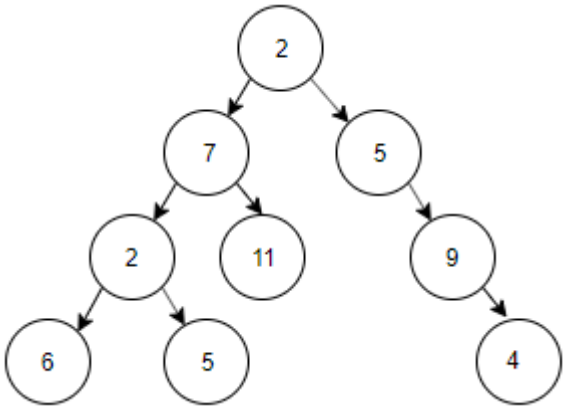
- Construct a binary tree by using postorder and inorder sequences given below.
  Inorder: N, M, P, O, Q
  Postorder: N, P, Q, O, M


M
N
O
P
Q

Construct a binary tree using inorder and level order traversal given below.
Inorder Traversal: 3, 4, 2, 1, 5, 8, 9
Level Order Traversal: 1, 4, 5, 9, 8, 2, 3

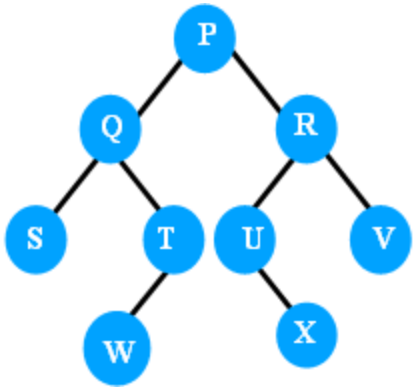

1
4
5
9
8
2
3

write the in-order traversal.



6, 2, 5, 7, 11, 2, 5, 9, 4

```
public void inorder(Tree root) {
inorder(root.left);
System.out.println(root.data);
inorder(root.right);
}
```



S W T Q X U V R P