Data Structures and Algorithms

Lecture 06

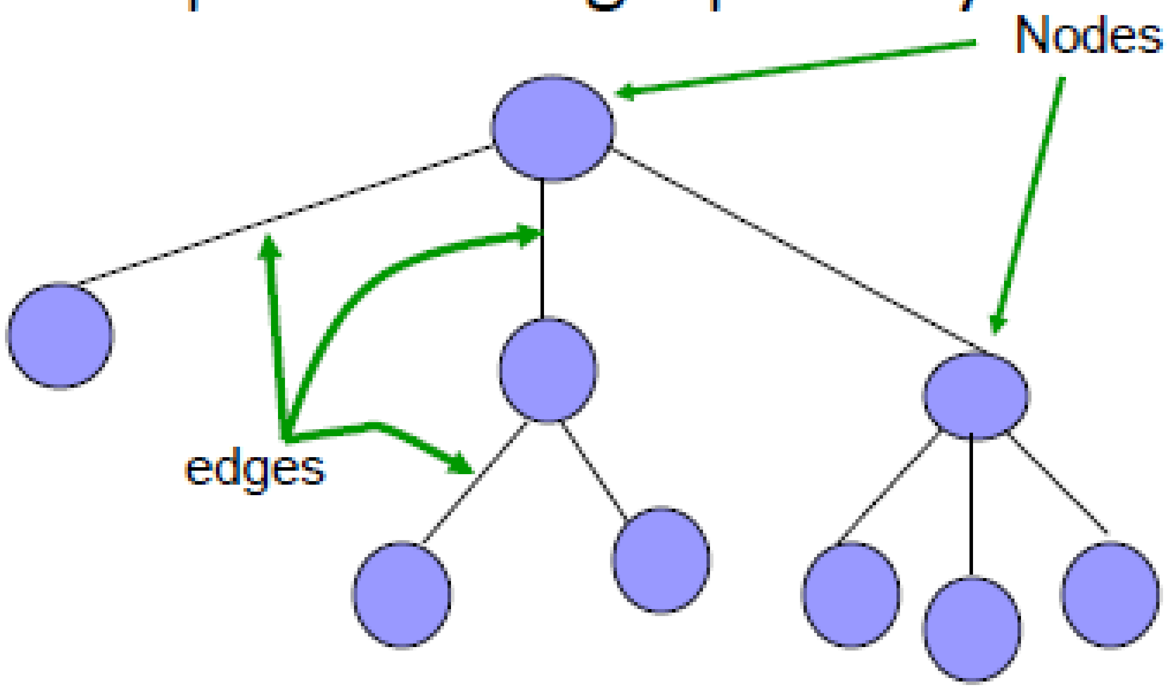
Road Map

- Introduction – Trees
- Trees Terminology
- Binary Tree
- Traversals

Introduction

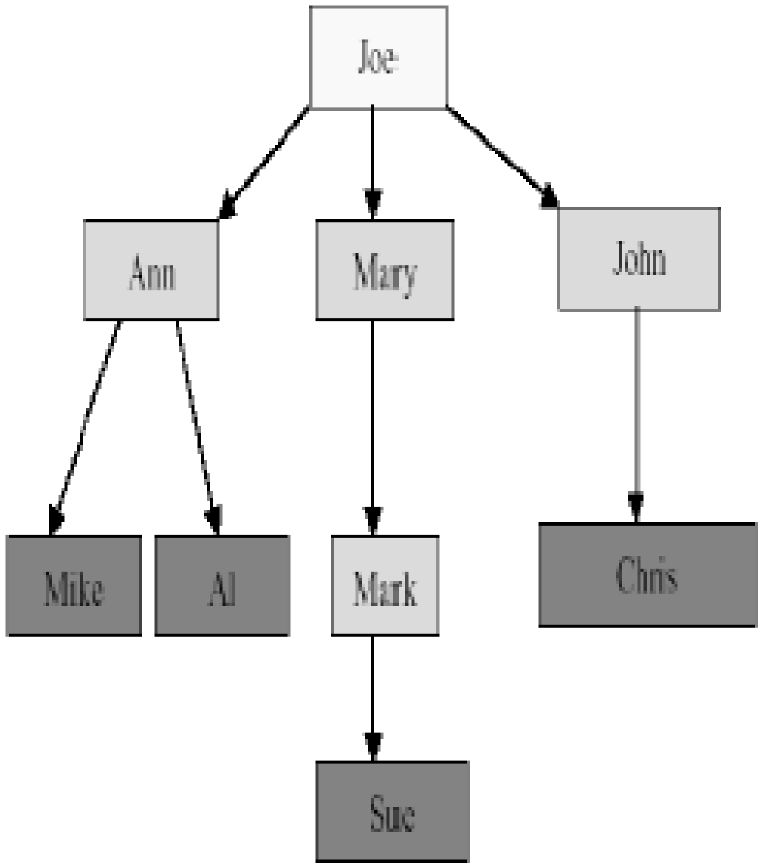
- Non- linear data structure
- Used to represent data, containing a hierarchical relationship among elements
  - University Structure
  - Table of Contents

- A tree consist of *nodes* connected by *edges*
- It can be represented graphically as follows:



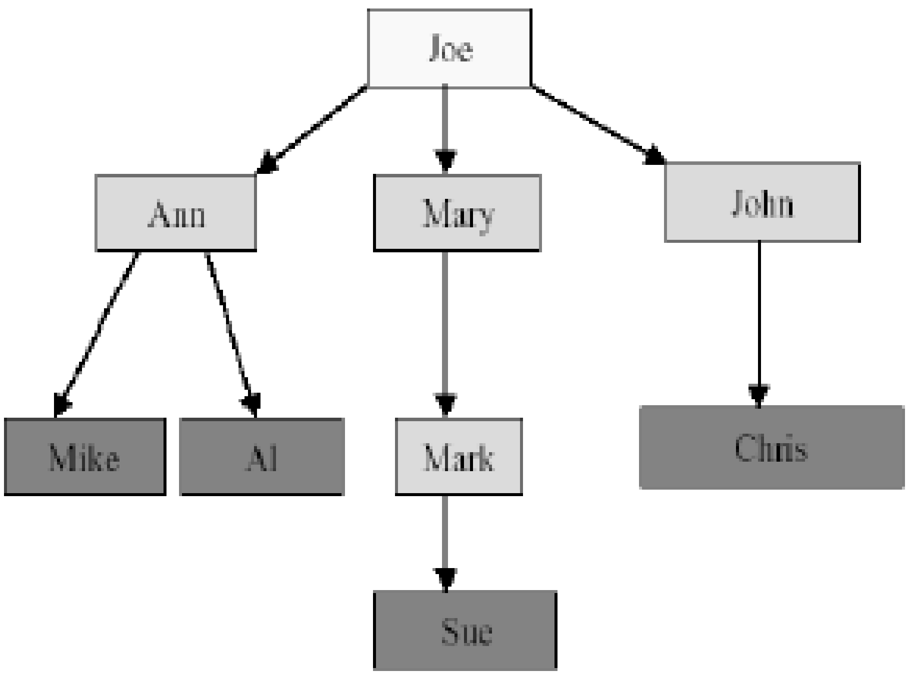
Terminology

- The element at the top of the hierarchy is the **root**.
- Elements next in the hierarchy are the **children** of the root.
- Elements next in the hierarchy are the **grandchildren** of the root and so on.
- Elements at the lowest level of the hierarchy are the **leaves**.



Some other definitions

- Leaves, Parent, Grandparent, Siblings, Ancestors, Descendents



Leaves = {Mike,Al,Sue,Chris}

Parent(Mary) = Joe

Grandparent(Sue) = Mary

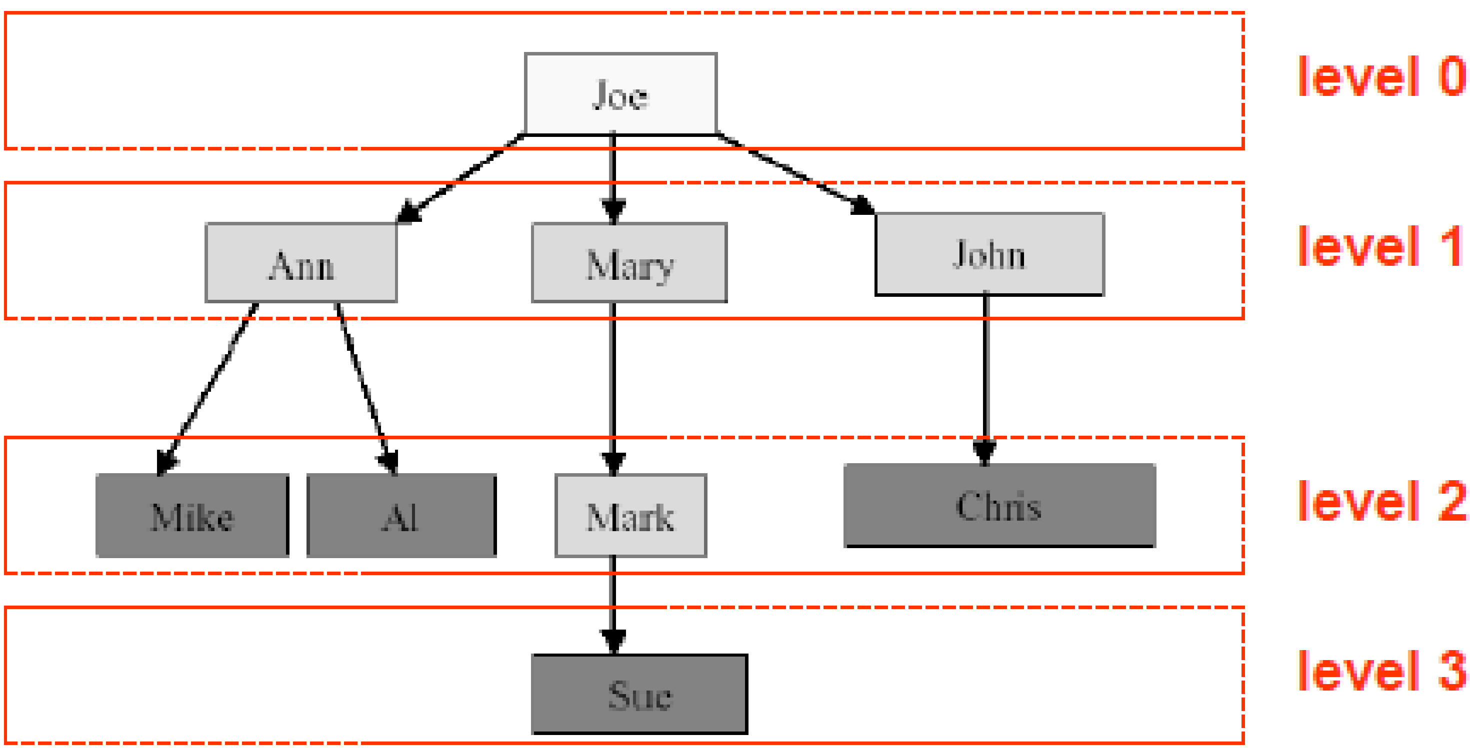
Siblings(Mary) = {Ann,John}

Ancestors(Mike) = {Ann,Joe}

Descendents(Mary)={Mark,Sue}

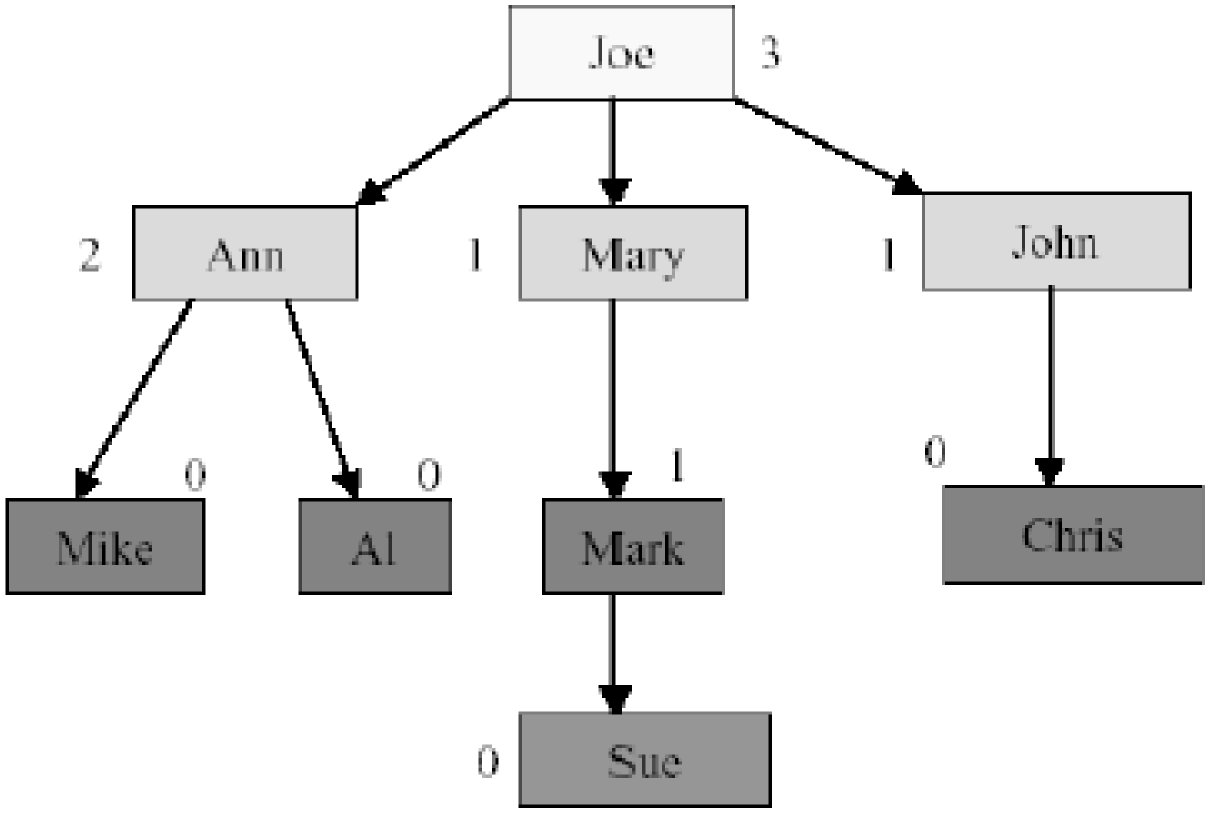
Define Levels

- Root is at level 0 and its children are at level 1



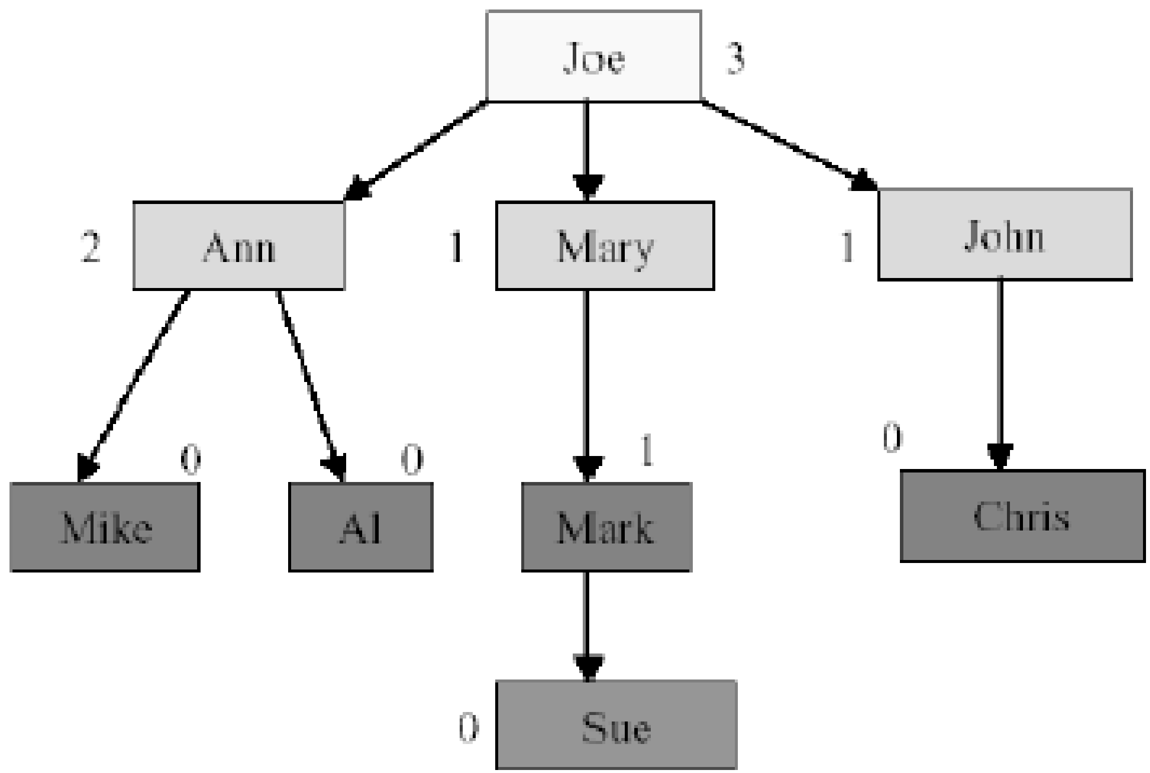
Node Degree

- Node degree is the number of children it has



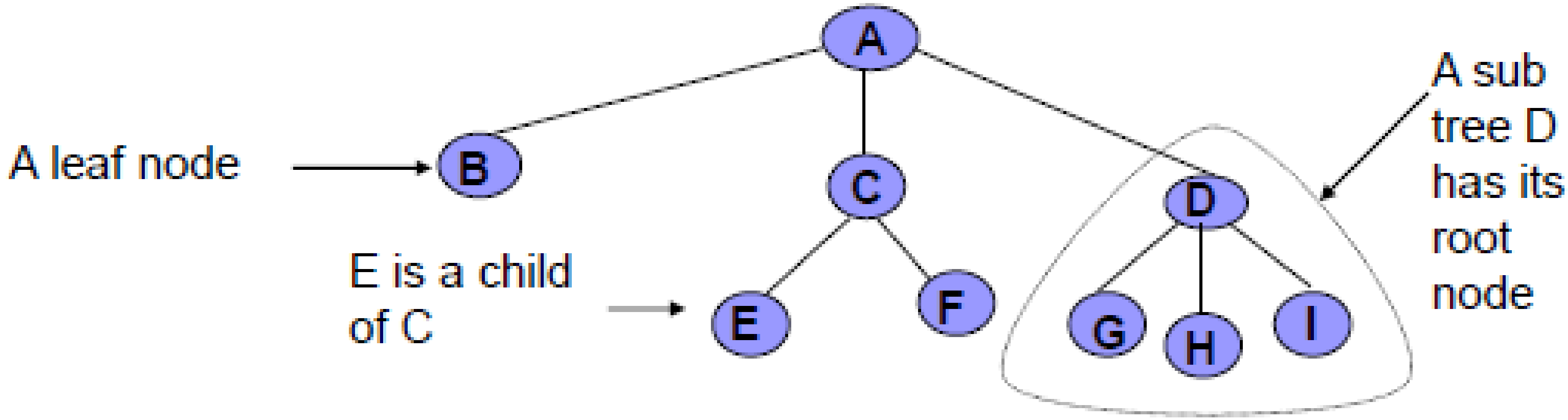
Tree Degree

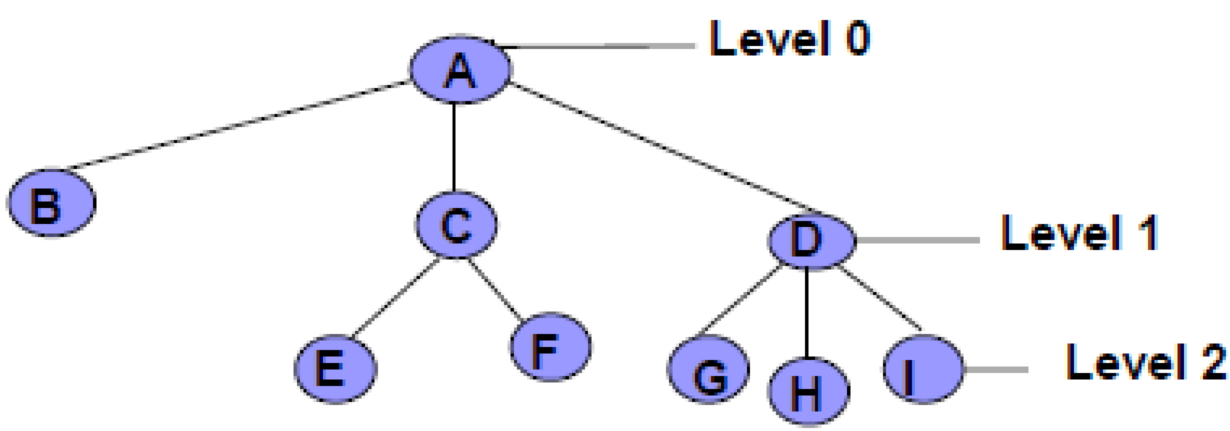
- Tree degree is the maximum of node degrees



tree degree = 3

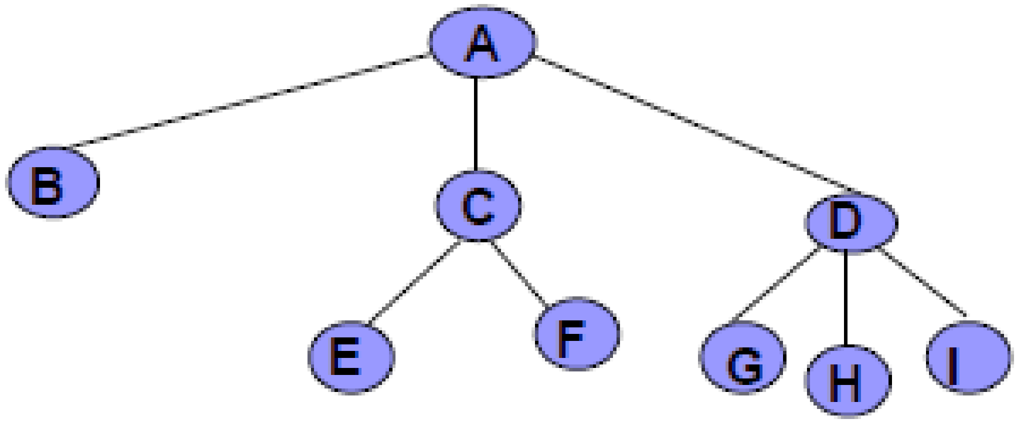
Sub tree – Any node may be considered to be the root of a *sub tree*, which consists of its children and its children's children, and so on.





Path

- If $n_1, n_2, \ldots n_k$ is a sequence of nodes in a tree such that n_i is the parent of n_{i+1}, for $1 \le i \le k$, then this sequence is called a *path* from node n_1 to n_k



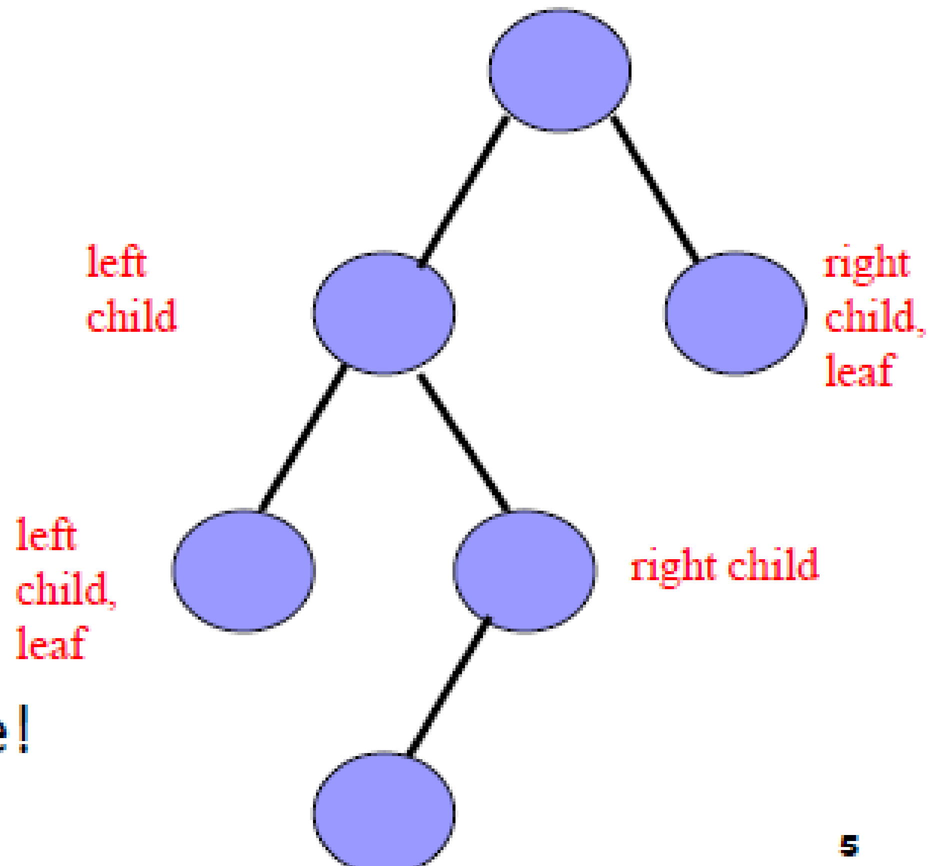
- Length of a path is one less than the number of nodes in a path.
- Node to Node(itself) – length of path is zero
- Height of a node – length of the longest path from node to a leaf.
- Height of a tree – the height of the root node.
- Depth of a node – length of the path from the root to that node.

Recursive Definition

1. A single node by itself is a tree. This node is also the root of this tree.
2. Let t_1, t_2, …, t_k be disjoint trees with roots r_1, r_2, …, r_k respectively, and let *R* be another node. We can get a new tree by making *R* the parent of the nodes r_1, r_2, …, r_k.

Binary Trees

- A tree in which each node can only have 0, 1, or 2 children.
- A binary tree is either *empty* or consists of a root, and (possibly empty) left and right subtrees.
  - Note the recursive definition in the above!

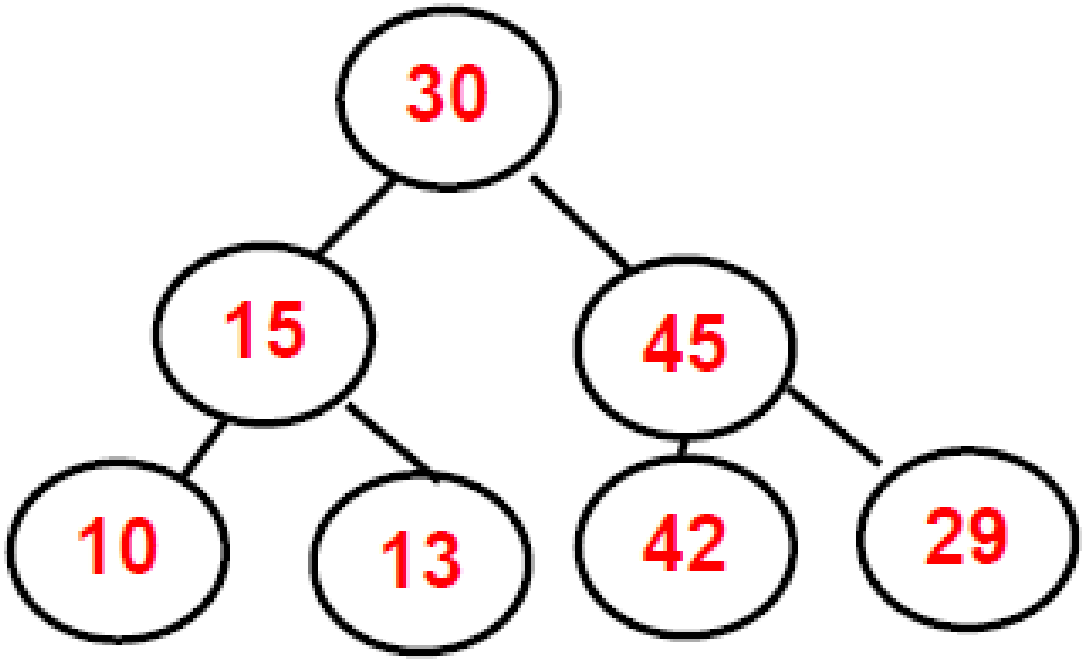


Tree Traversal

- Methods of visiting(processing) each node in the tree exactly one time.

- Breadth First
- Depth First

Breadth First Traversal



30, 15, 45, 10, 13, 42, 29

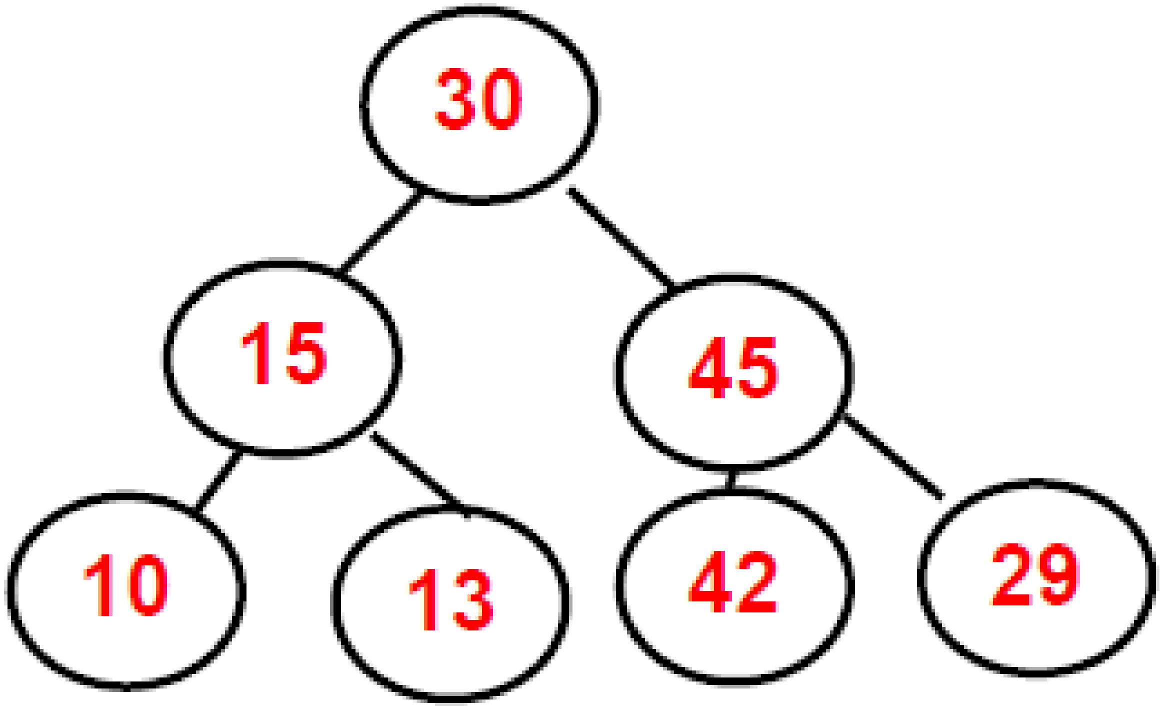
Algorithm (Breadth First – Level by Level)

```
Algorithm breadthFirst (root)
Process tree using breadth-first traversal.
   Pre     root is node to be processed
   Post    tree has been processed
 1 set currentNode to root
 2 createQueue (bfQueue)
 3 loop (currentNode not null)
   1  process (currentNode)
   2  if (left subtree not null)
      1  enqueue (bfQueue, left subtree)
   3  end if
   4  if (right subtree not null)
      1  enqueue (bfQueue, right subtree)
   5  end if
   6  if (not emptyQueue(bfQueue))
      1  set currentNode to dequeue (bfQueue)
   7  else
      1  set currentNode to null
   8  end if
 4 end loop
 5 destroyQueue (bfQueue)
end breadthFirst
```

Depth First Traversal

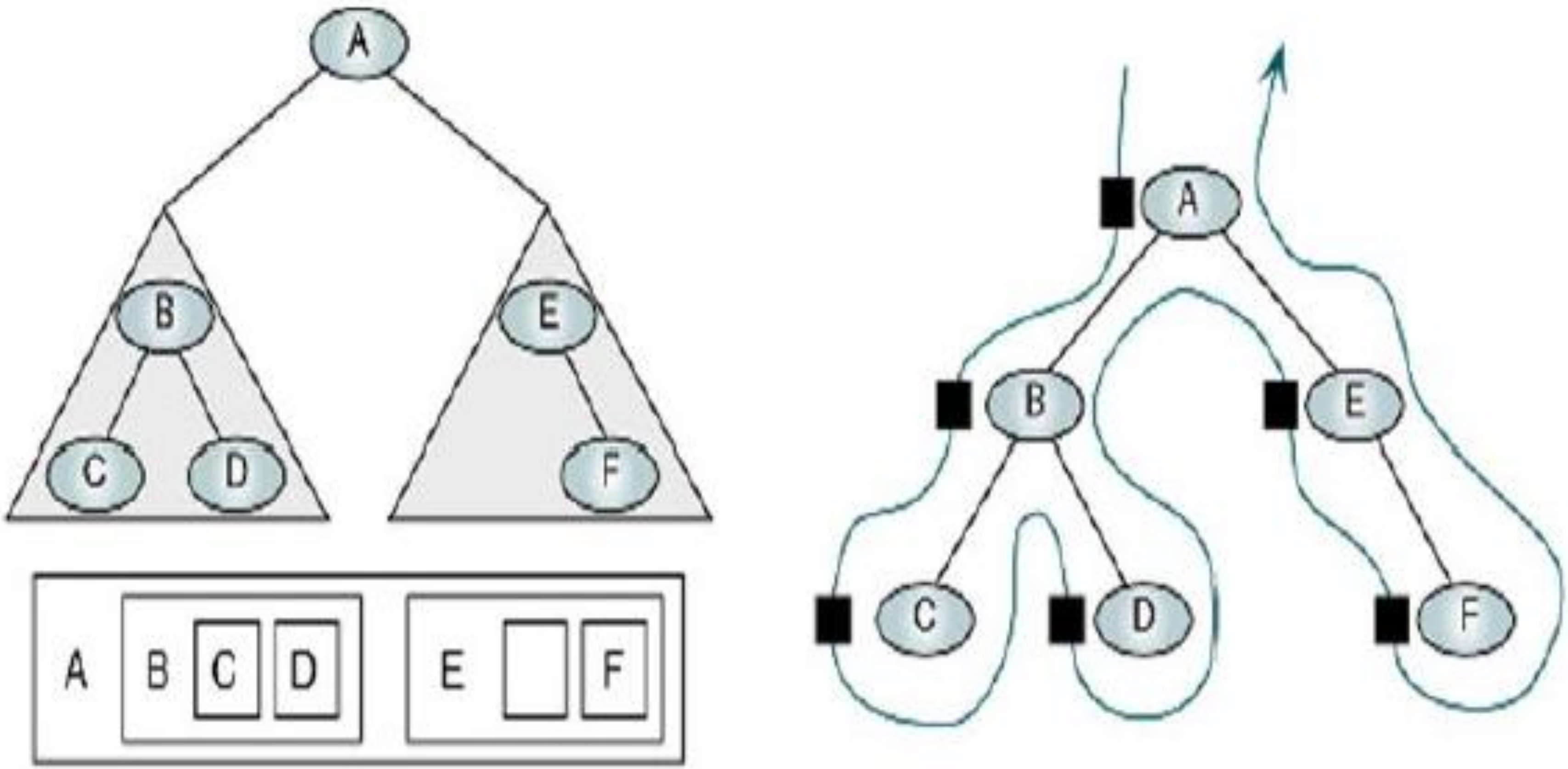
- Preorder
- Inorder
- Postorder

Preorder Tree Traversal



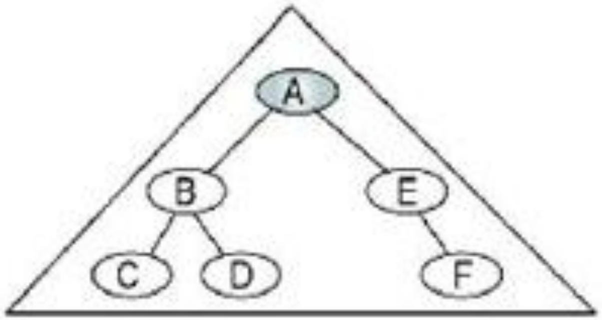
30, 15, 10, 13, 45, 42, 29

```
Algorithm preOrder (root)
Traverse a binary tree in node-left-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
 1 if (root is not null)
   1  process (root)
   2  preOrder (leftSubtree)
   3  preOrder (rightSubtree)
 2 end if
end preOrder
```

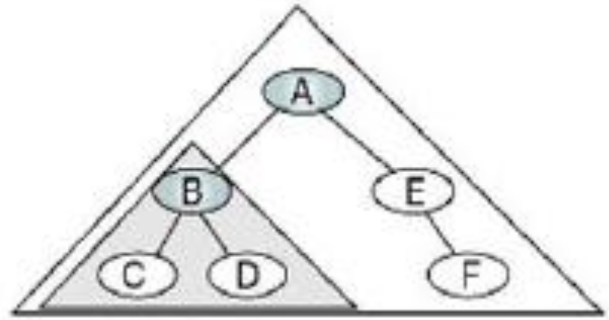


(a) Processing order

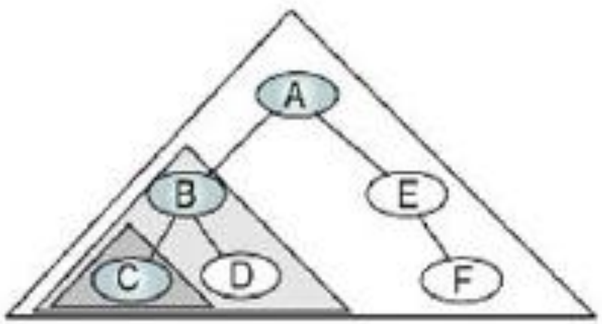
(b) "Walking" order



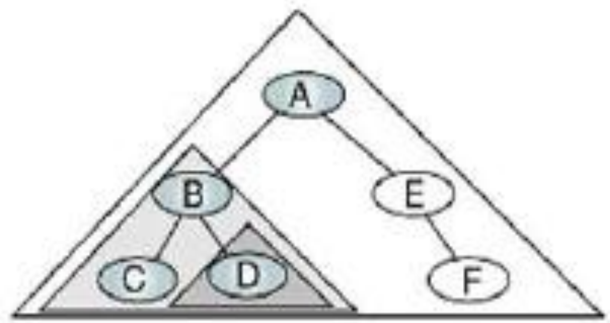
(a) Process tree A



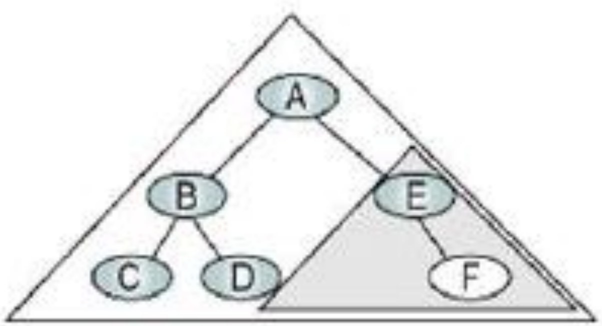
(b) Process tree B



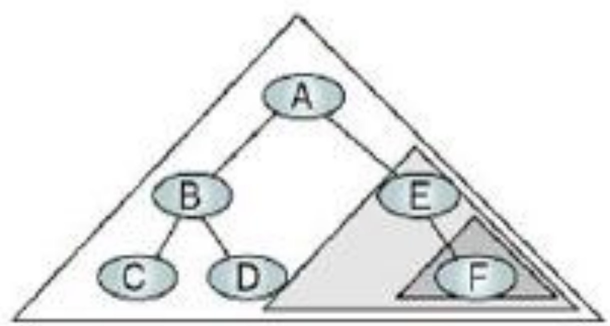
(c) Process tree C



(d) Process tree D

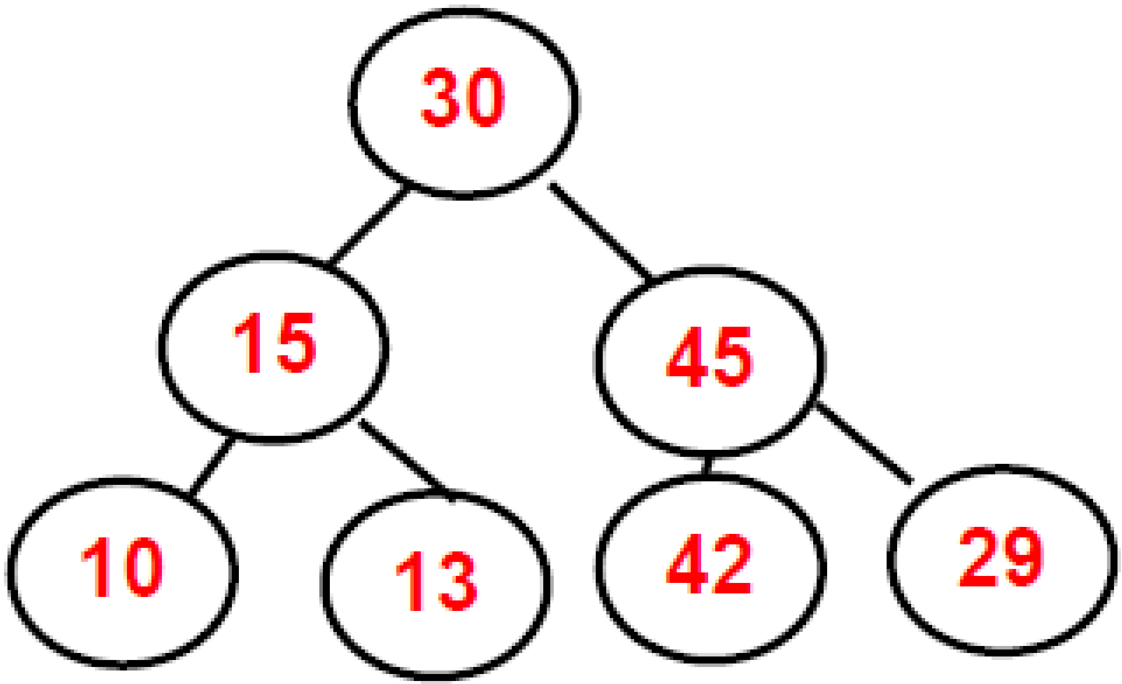


(e) Process tree E



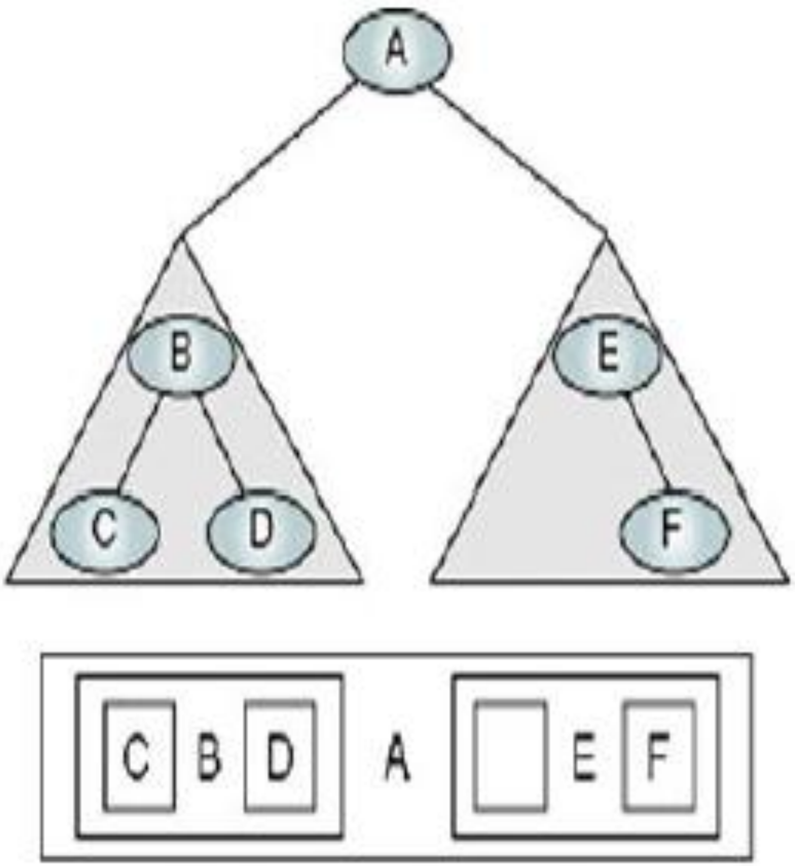
(f) Process tree F

Inorder Tree Traversal

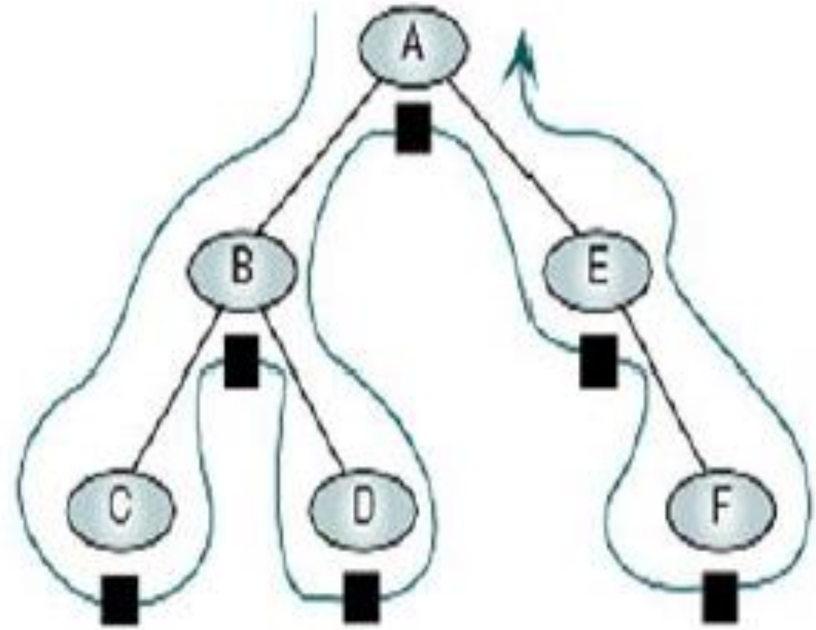


10, 15, 13, 30, 42, 45, 29

```
Algorithm inOrder (root)
Traverse a binary tree in left-node-right sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
1 if (root is not null)
   1  inOrder (leftSubTree)
   2  process (root)
   3  inOrder (rightSubTree)
2 end if
end inOrder
```

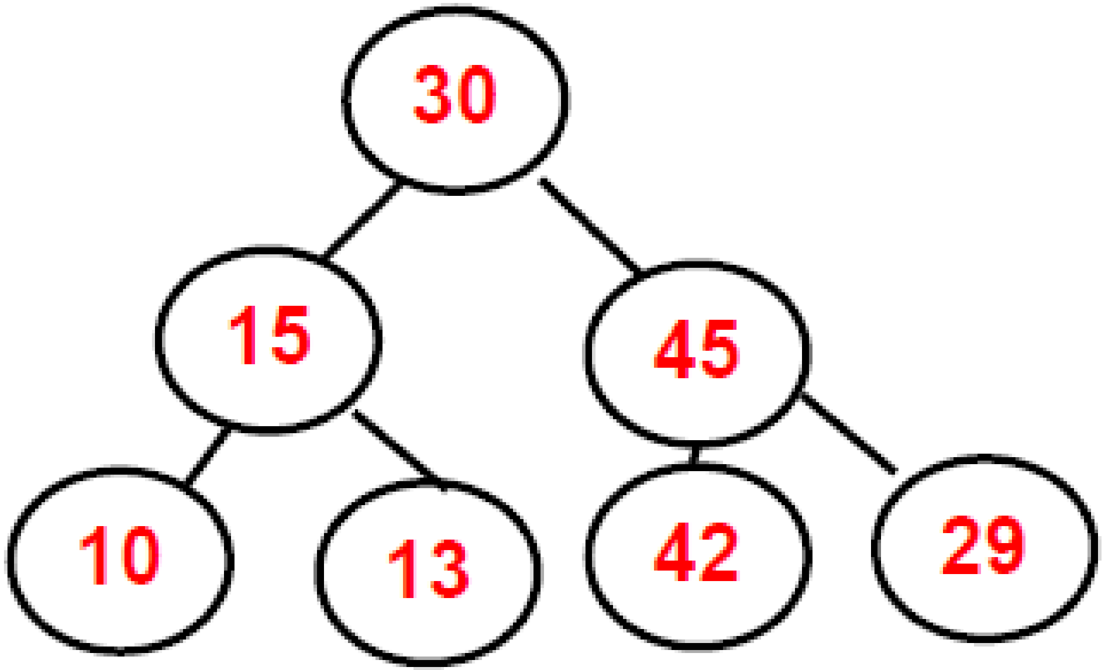


(a) Processing order



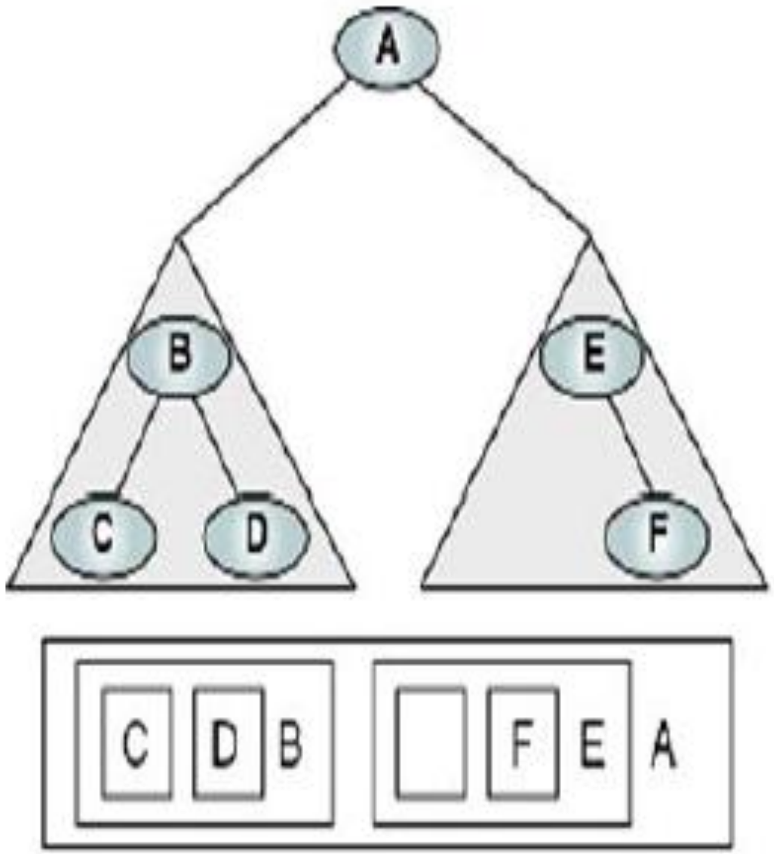
(b) "Walking" order

Postorder Tree Traversal

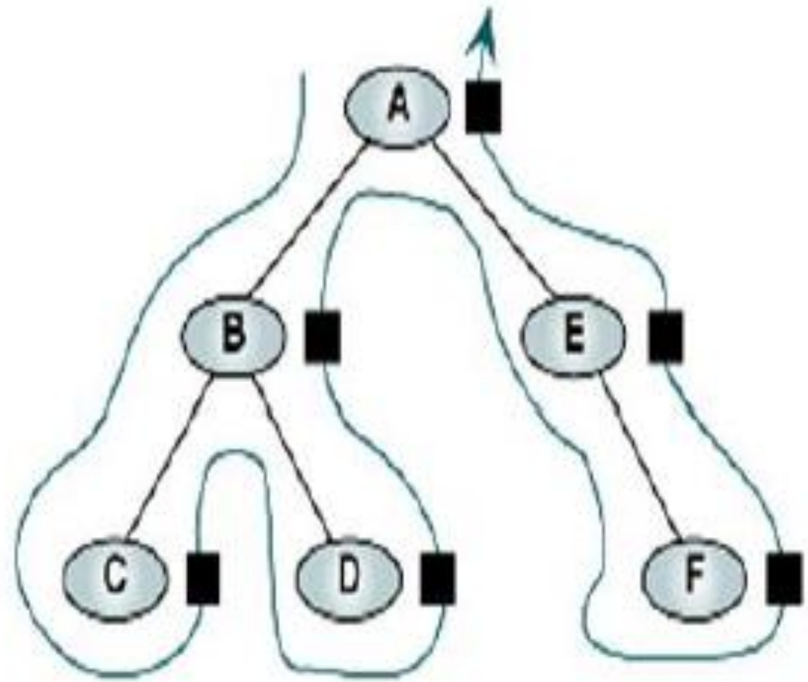


10, 13, 15, 42, 29, 45, 30

```
Algorithm postOrder (root)
Traverse a binary tree in left-right-node sequence.
   Pre  root is the entry node of a tree or subtree
   Post each node has been processed in order
 1 if (root is not null)
   1   postOrder (left subtree)
   2   postOrder (right subtree)
   3   process (root)
 2 end if
end postOrder
```



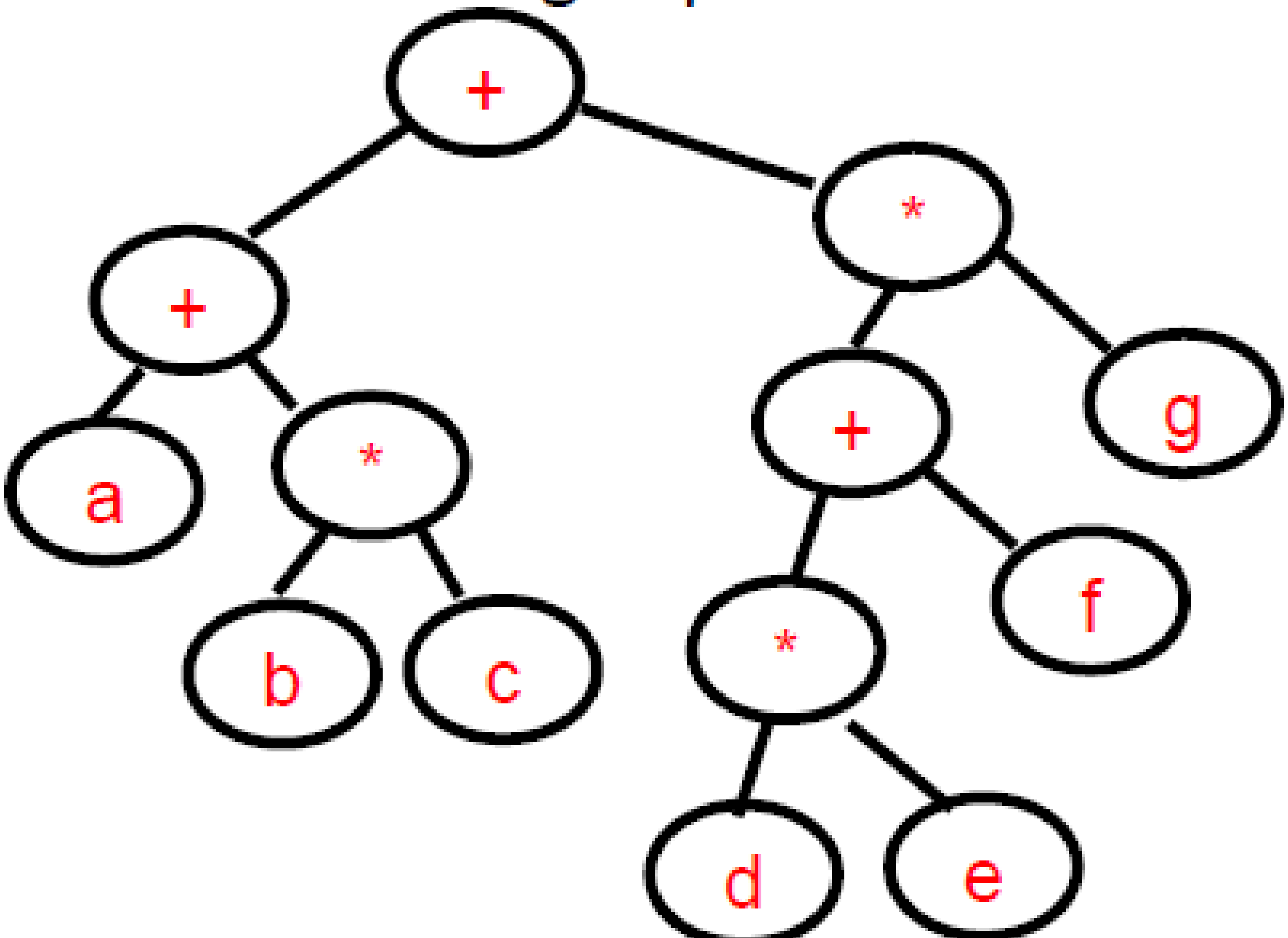
(a) Processing order



(b) "Walking" order

Exercise

- Consider the following expression tree:



- Write the path order for the following
  - Pre-order
  - In-order
  - Post-order

Questions ??