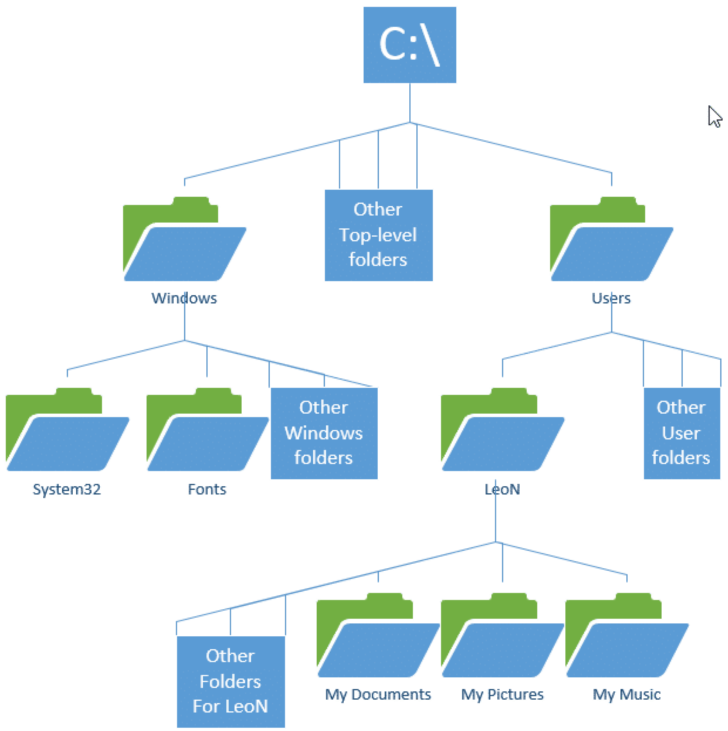
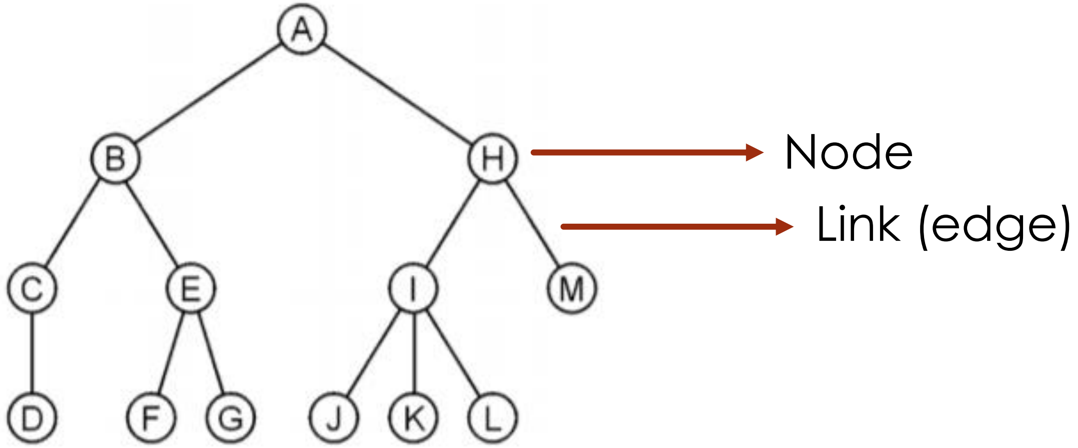
Trees

Trees

- A tree is an ADT that stores elements **hierarchically**.
- A tree T is a set of nodes storing elements in a parent-child relationship with the following properties:
  - T has a special node r , called the root of T .
  - Each node v of T different from r has a parent node u.
- Direct applications:
  - Organizational charts
  - File systems
  - Programming environments

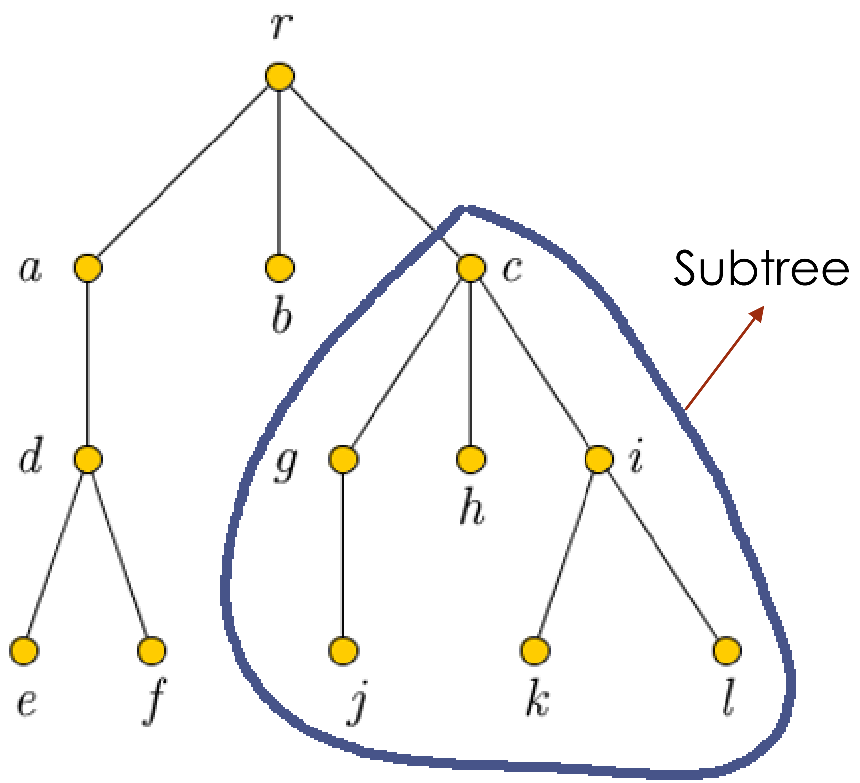




Tree Terminologies

- If node u is the **parent** of node v, then we say that v is a **child** of u.
- Two nodes that are children of the same parent are **siblings** .
- A node is **external** (**leaf**) if it has no children, and it is **internal** if it has one or more children.
- The **ancestors** of a vertex are the vertices in the path from the root to this vertex.
- The **descendants** of a vertex v are those vertices that have v as an ancestor.
- **Depth** : The depth of a node is the number of edges from the node to the tree's root node. In other words, the depth of v is the number of ancestors of v.
- The **height** of a tree T is equal to the maximum depth of an external node of T.
- **Height of a node v** is the number of edges on the *longest path* from v to a leaf. A leaf node will have a height of 0. The height of a tree is the largest level of the vertices of a tree which is he height of a root.
- A **subtree** of a **tree** T is a **tree** S consisting of a node in T and all of its descendants in T.

Example



- The parent of d is a.
- The children of c are $g, h,$ and i .
- The siblings of g are h and i.
- The ancestors of f are $d, a,$ and r.
- The descendants of a are $d, e,$ and f.
- The internal vertices are $r, a, d, c, g,$ and i .
- The leaves are $e, f, b, j, h, k,$ and l.
- The height of d is 1.
- The height of c is 2.
- The height of b is 0.
- The height of r is 3 which is the height of tree.
- The depth of d is 2.
- The depth of r is 0.
- The depth of k is 3.
- The height of Tree is 3.

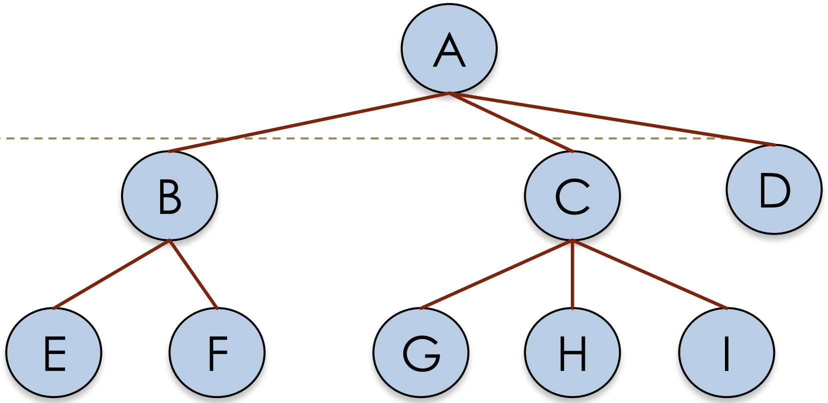
Theorem: A tree with n nodes has n − 1 edges.

Tree Traversal

- A traversal of a tree T is a systematic way of accessing, or "visiting," all the nodes of T.

- There are three main types of tree traversals:
  - Preorder: A node is visited **before** its descendants.
  - Postorder: a node is visited **after** its descendants.
  - Inorder: We will talk about this later. This is only supported in binary tree.

Tree Traversal



- preorder: a node is visited before its descendants

$O(n)$

Algorithm *preOrder*(*v*)
    visit(*v*)
    for each child *w* of *v*
        preOrder (*w*)

preorder(A) visits: ABEFCGHID

- postorder: a node is visited after its descendants

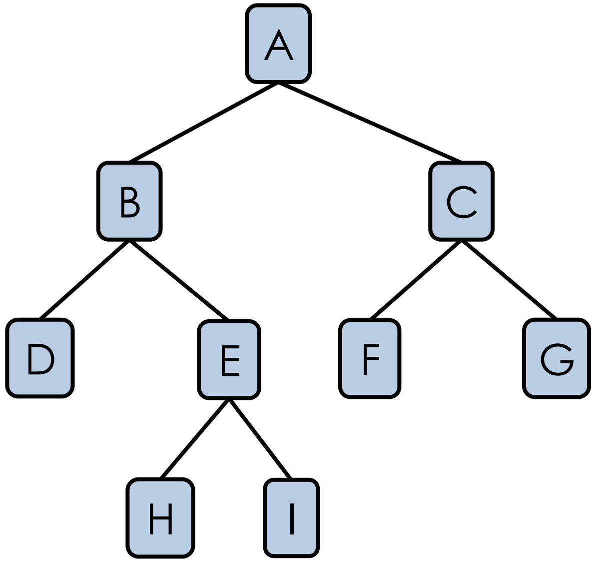
$O(n)$

Algorithm *postOrder*(*v*)
    for each child *w* of *v*
        postOrder (*w*)
    visit(*v*)

postorder(A) visits: EFBGHICDA

Binary Trees

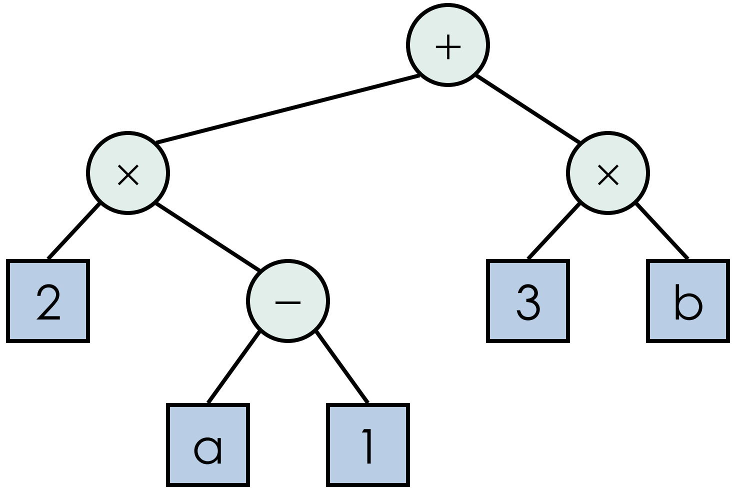
- A binary tree is an ordered tree with the following properties:
  - Each internal node has only two children
  - The children of a node are an ordered pair (**left child**, **right child**)
- Recursive definition: a binary tree is
  - A single node is a binary tree
  - Two binary trees connected by a root is a binary tree
- Applications:
  - arithmetic expressions
  - decision processes
  - searching



Arithmetic Expression Tree

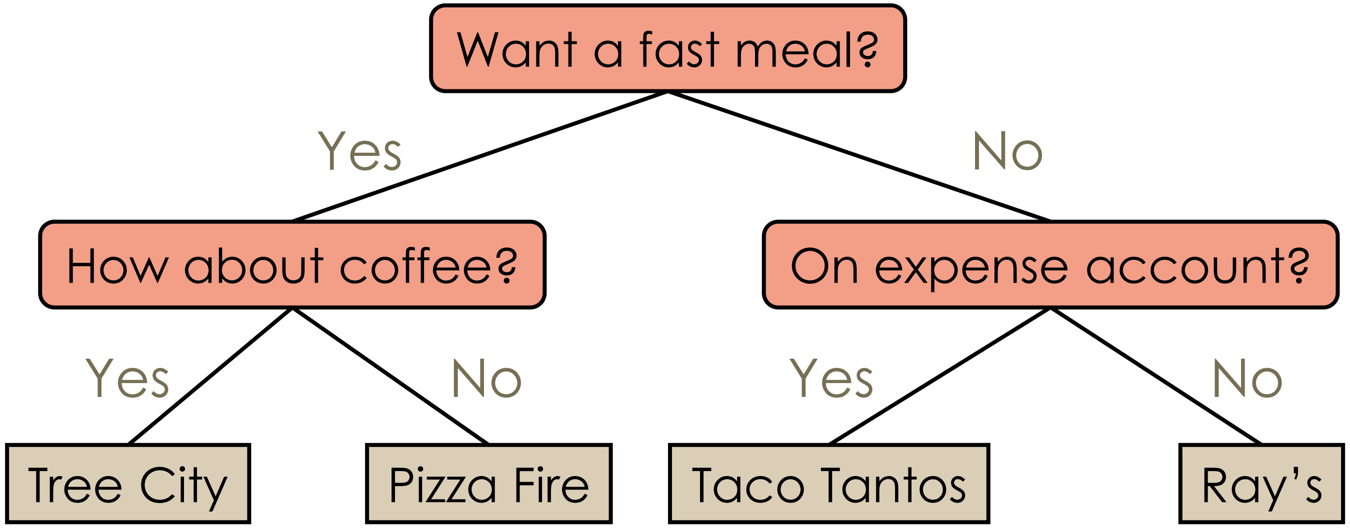
- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands

- Ex: arithmetic expression tree for expression $(2 \times (a - 1) + (3 \times b))$



Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Ex: dining decision

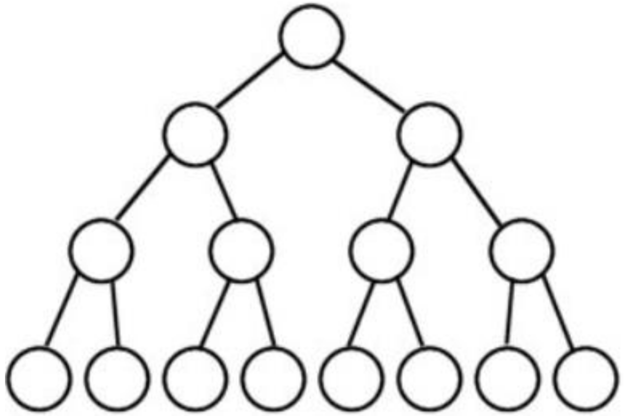


Binary Tree Types

- Two main Types:
  - Full Binary tree
  - Complete Binary Tree

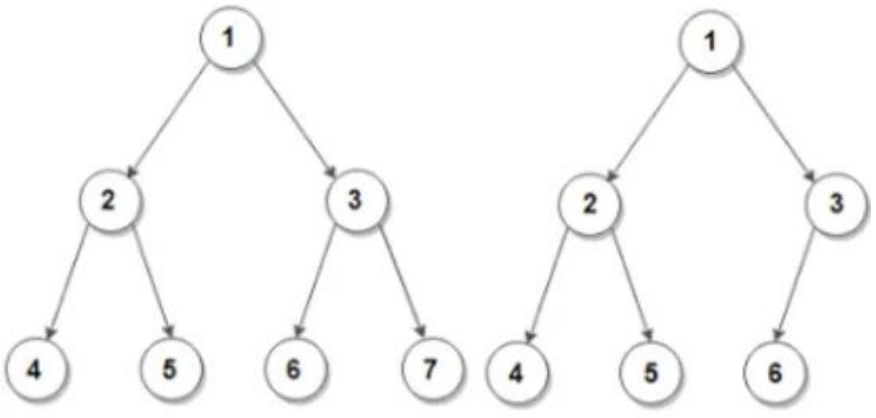
Full Binary Tree

A full binary tree is a tree in which every node other than the leaves has two children.

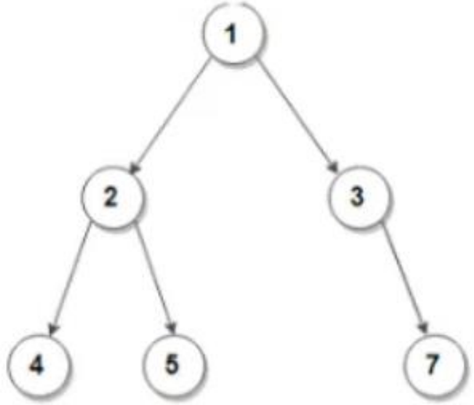


Complete Binary Tree

- A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.



Complete Binary Tree



Not a Complete Binary Tree

Number of nodes at Levels

- Level l has at most 2^l nodes
- The number of external nodes in T is at least h + 1 and at most 2^h .

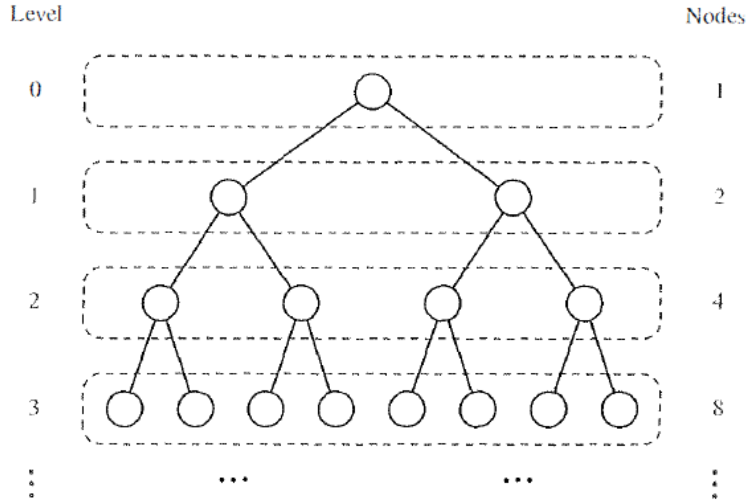


Figure 2.25: Maximum number of nodes in the levels of a binary tree.

Binary Tree Traversals

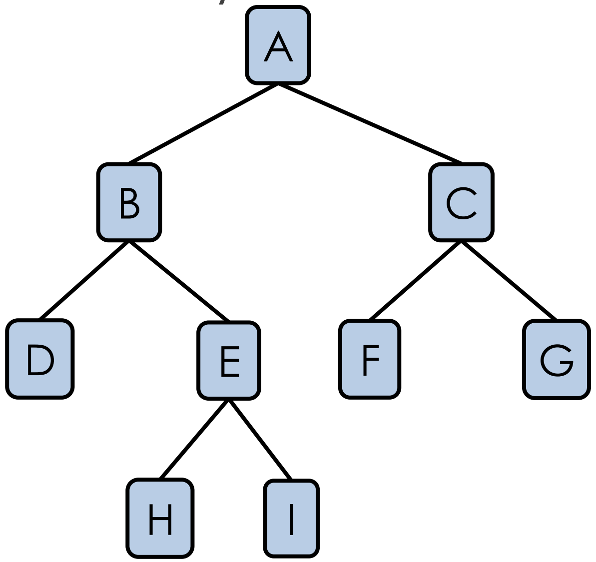
- Three main types:
  1) Preorder traversal : Preorder (Root, Left, Right)
     - the root node is visited first, then the left subtree and finally the right subtree.
  2) Postorder Traversal: Postorder (Left, Right, Root)
     - the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.
  3) Inorder Traversal: Inorder (Left, Root, Right)
     - the left subtree is visited first, then the root and later the right sub-tree.

Preorder Traversal of a Binary Tree

- Preorder traversal: Preorder (Root, Left, Right)
  1. the root node is visited first,
  2. Traverse the left subtree, i.e., call Preorder(left-subtree)
  3. Traverse the right subtree, i.e., call Preorder(right-subtree)

***O*(*n*)**

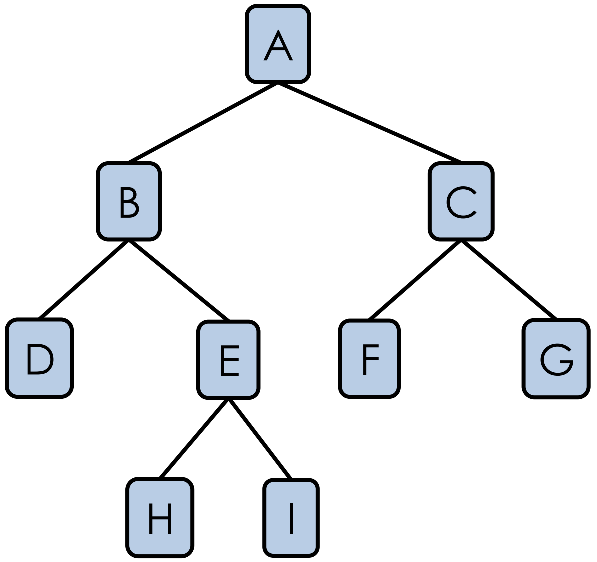
```
preorder (v)
    if x != Null
        print (x.value)
        preorder(x.leftchild)
        preorder(x.righchild)
```



Ex: ABDEHICFG

Postorder Traversal of a Binary Tree

- Postorder traversal: Postorder (Left, Right, Root)
  1. Traverse the left subtree, i.e., call Postorder(left-subtree)
  2. Traverse the right subtree, i.e., call Postorder(right-subtree)
  3. Visit the root.



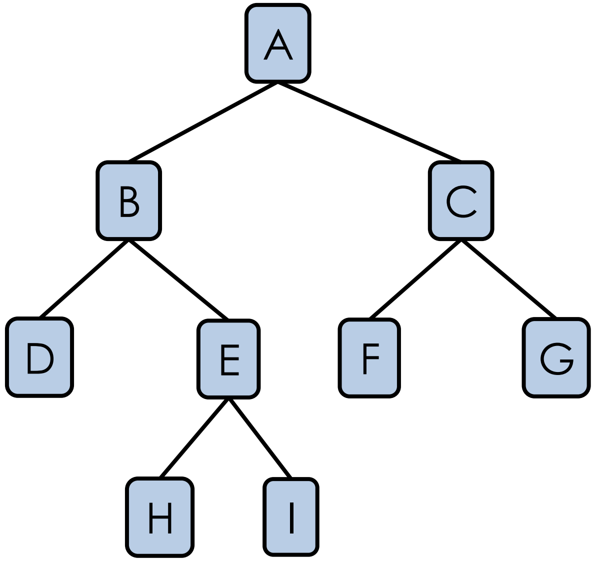
Ex: DHIEBFGCA

$O(n)$

```
postorder (v)
    if x! = Null
        postorder(x.leftchild)
        postorder(x.righchild)
        print (x.value)
```

Inorder Traversal of a Binary Tree

- Inorder traversal: Inorder (Left, Root, Right)
  1. Traverse the left subtree, i.e., call Inorder(left-subtree)
  2. Visit the root.
  3. Traverse the right subtree, i.e., call Inorder(right-subtree)



Ex: DBHEIAFCG

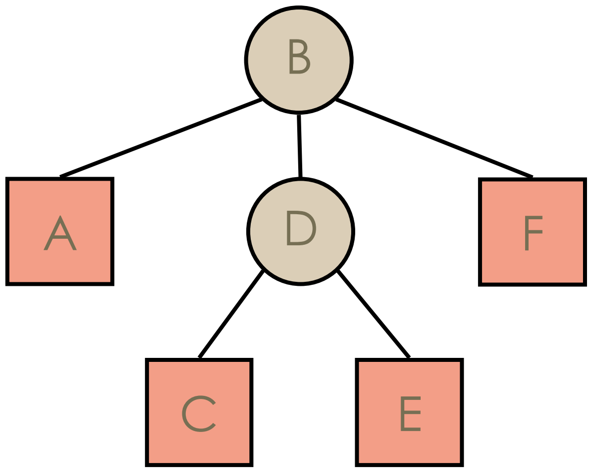
***O*(*n*)**

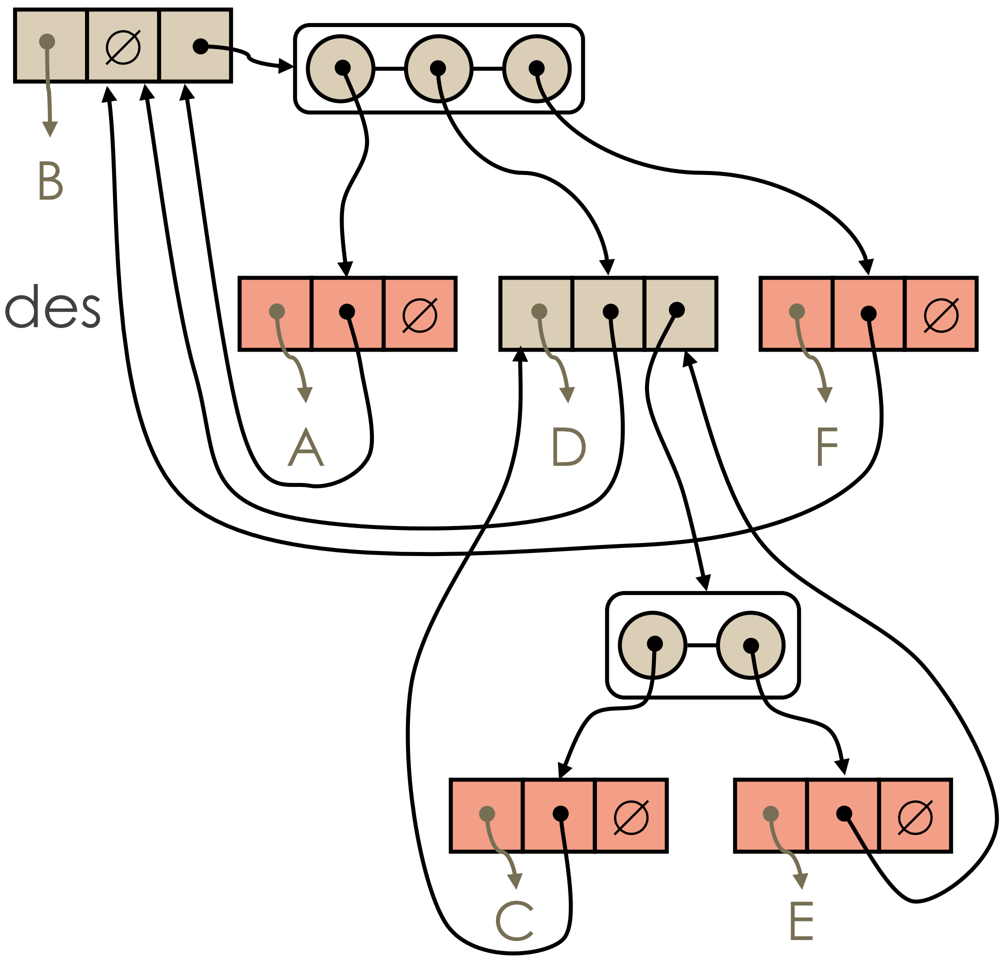
```
Inorder (v)
    if x != Null
        Inorder(x.leftchild)
        print (x.value)
        Inorder(x.righchild)
```

Linked Data Structure for Representing Trees

A node stores:

- element
- parent node
- sequence of children nodes

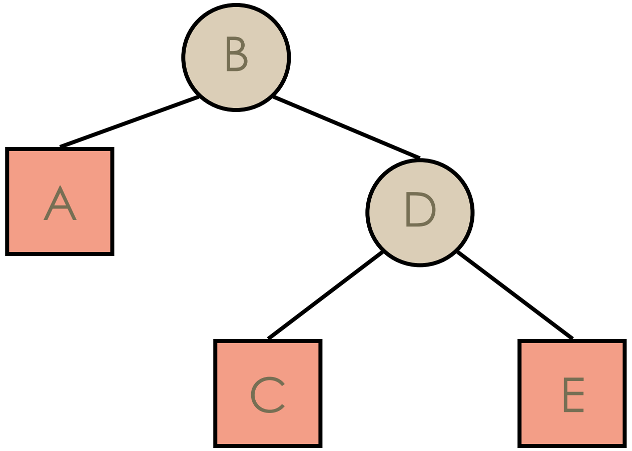


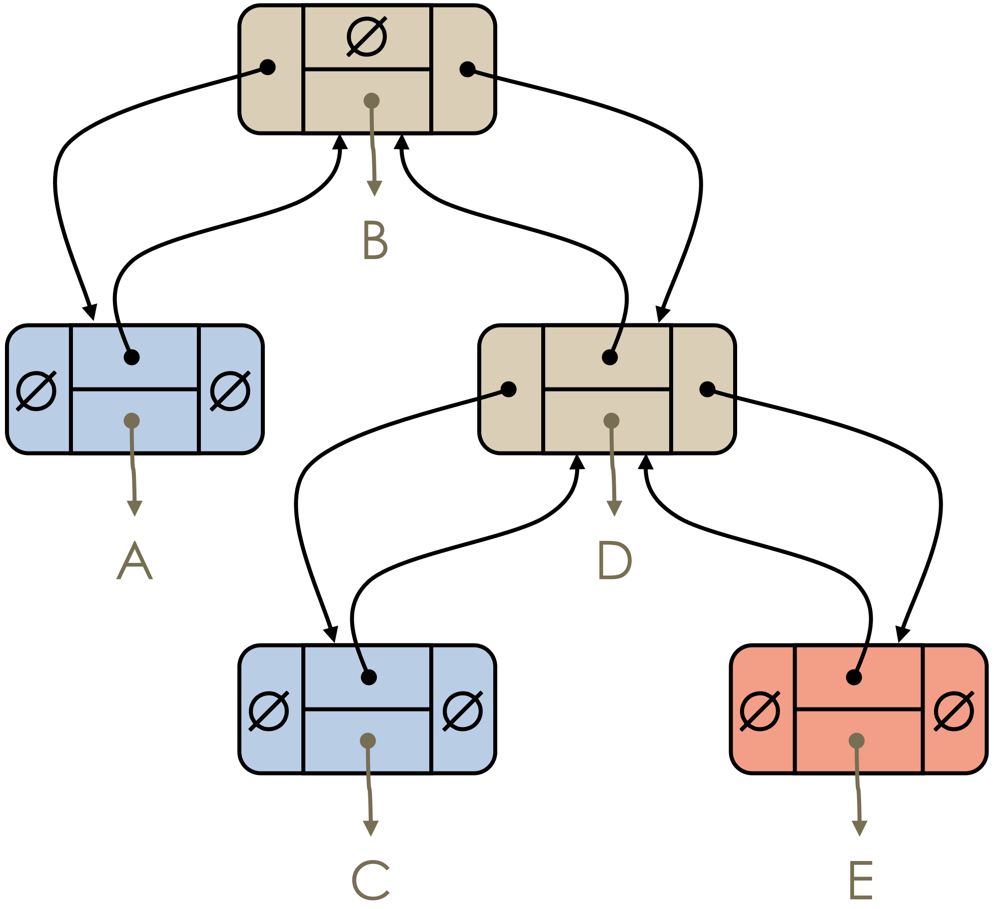


Linked Data Structure for Binary Trees

A node stores:

- element
- parent node
- left node
- right node





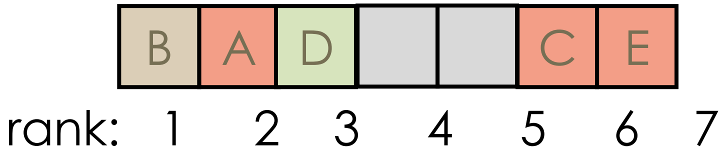
Array-Based Representation of Binary Trees

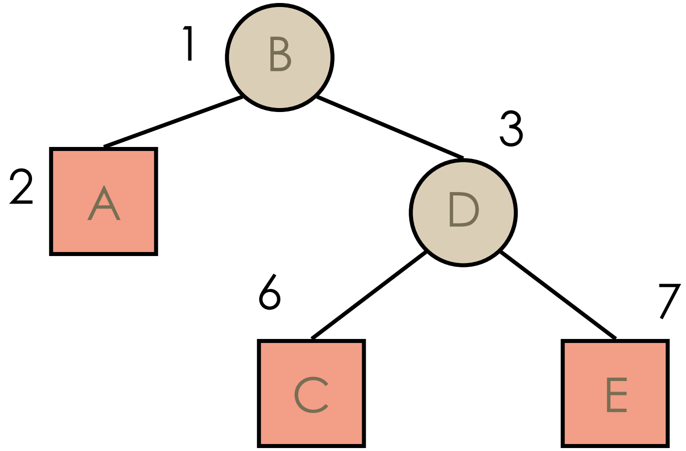
Nodes are stored in an array

- *rank*(root) = 1
- If *rank*(node) = *i*, then

  rank(leftChild) = 2**i*

  rank(rightChild) = 2**i* + 1





Ex: ‘A’ is left child of B

rank(A) = 2 * *rank*(B)

= 2 * 1 = 1

Ex: ‘E’ is right child of D

rank(E) = 2 * *rank*(D) + 1

= 2 * 3 + 1

= 7

Exercises

- **Write the iterative** Implementation (Pseudocode) of preorder and postorder traversals?
- The number of edges from the node to the deepest leaf is called __________ of the tree.
  a) Height
  b) Depth
  c) Length
  d) Width