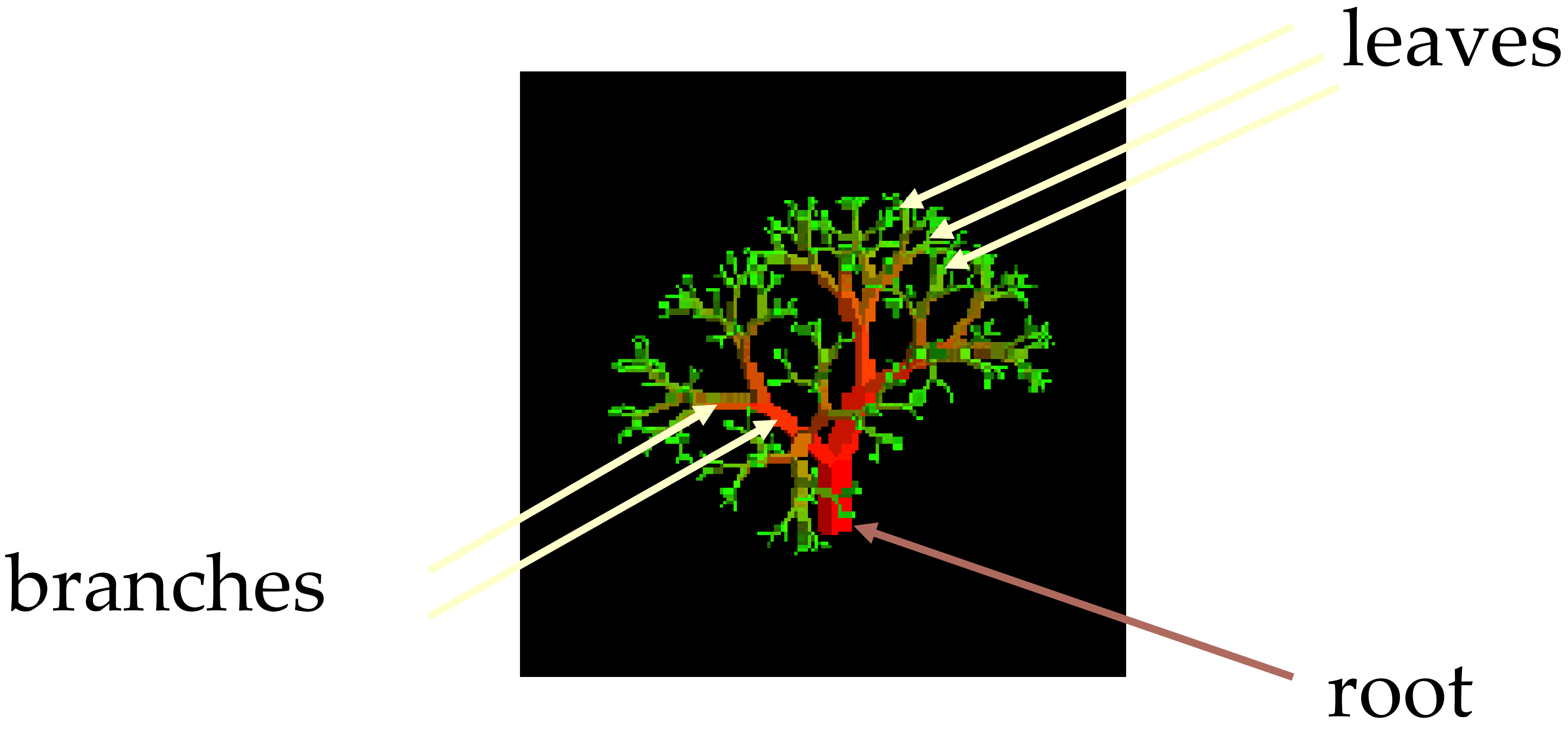
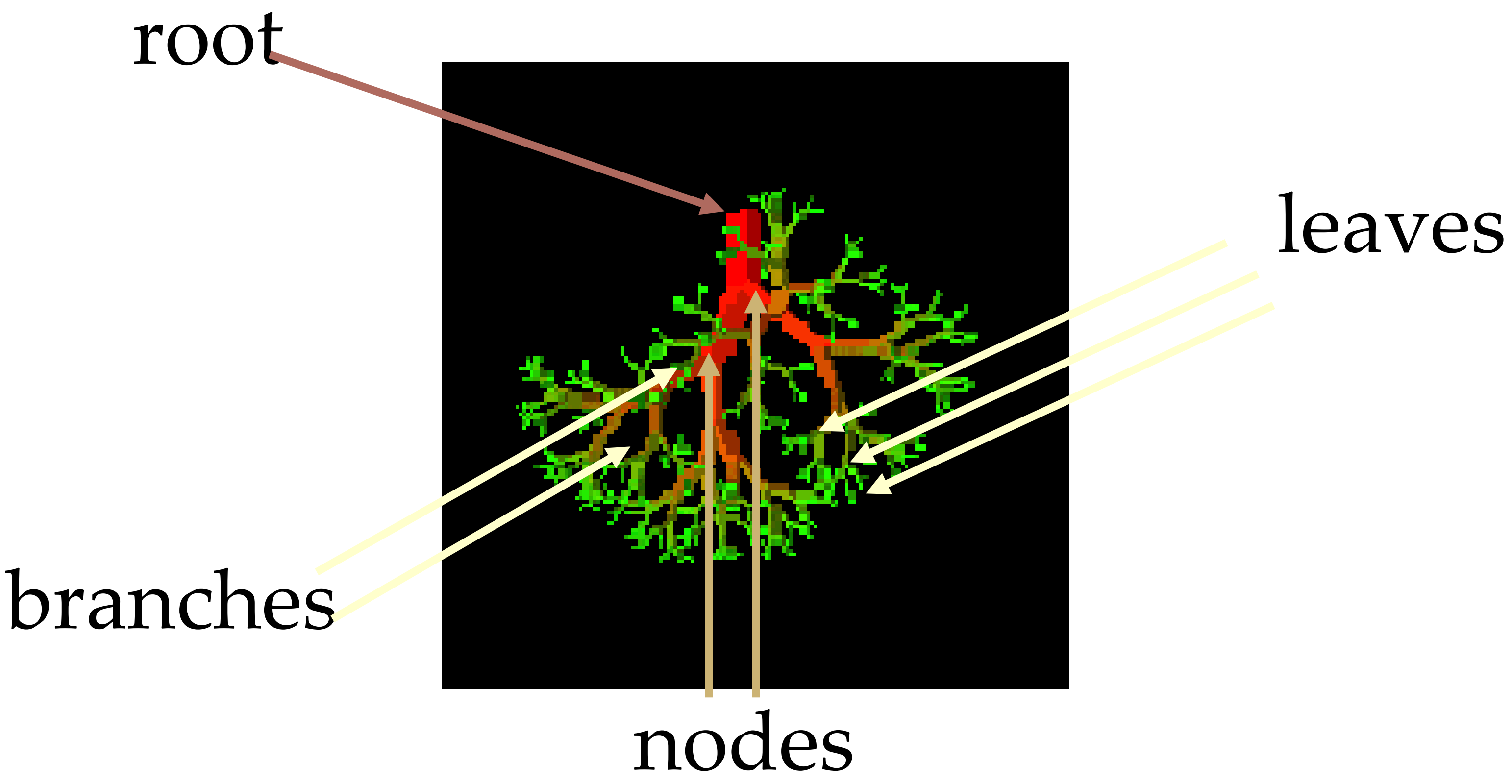
Introduction to Trees

Nature View of a Tree



Computer Scientist's View



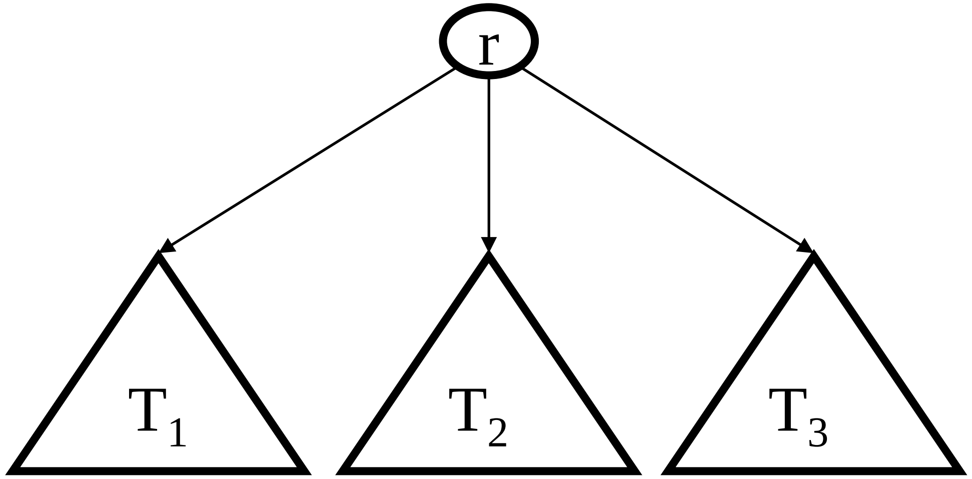
Definition of Tree

- A tree is a finite set of one or more nodes such that:
  - There is a specially designated node called the root.
  - The remaining nodes are partitioned into $n>=0$ disjoint sets $T_1, ..., T_n$, where each of these sets is a tree.
  - We call $T_1, ..., T_n$ the subtrees of the root.

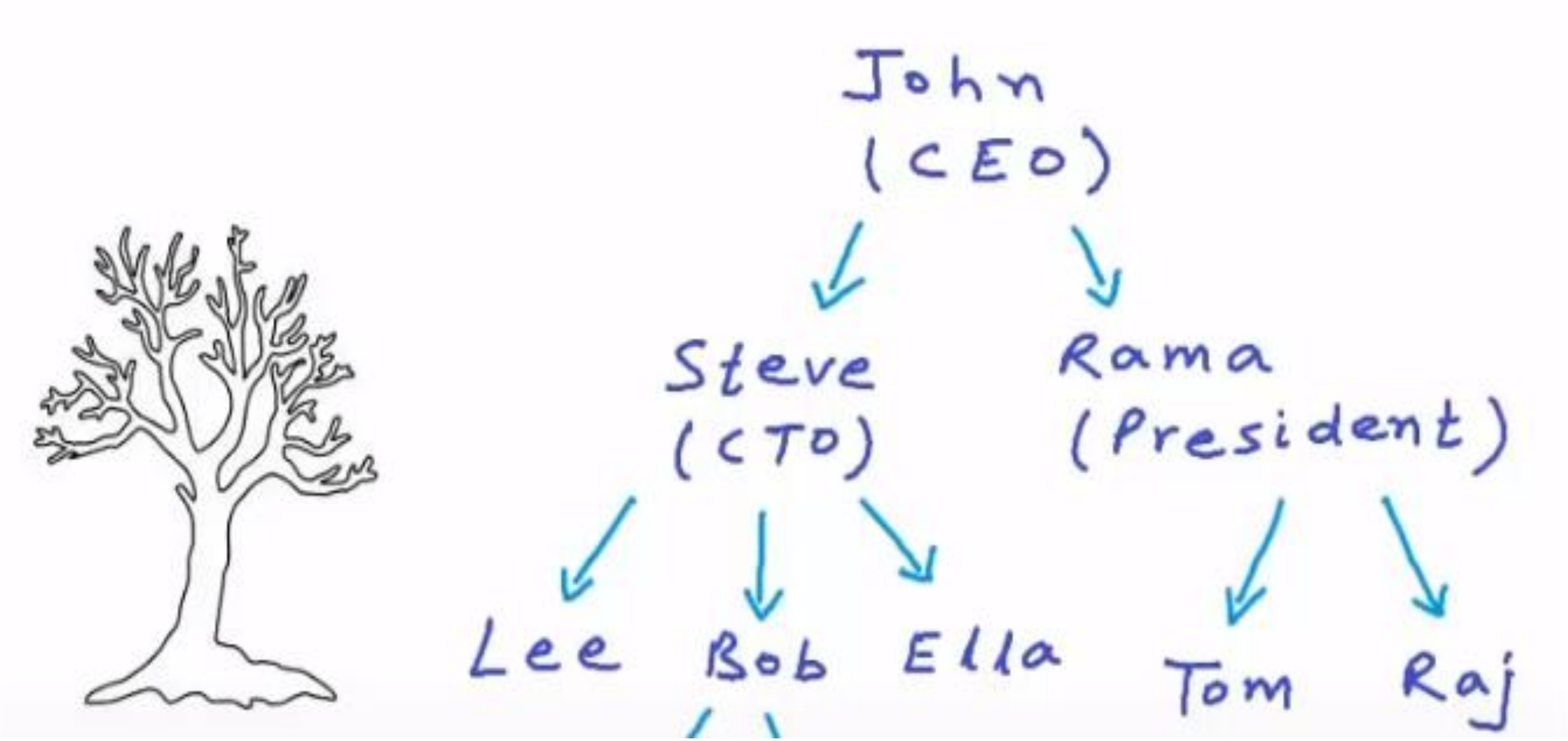
Recursive definition of a Tree

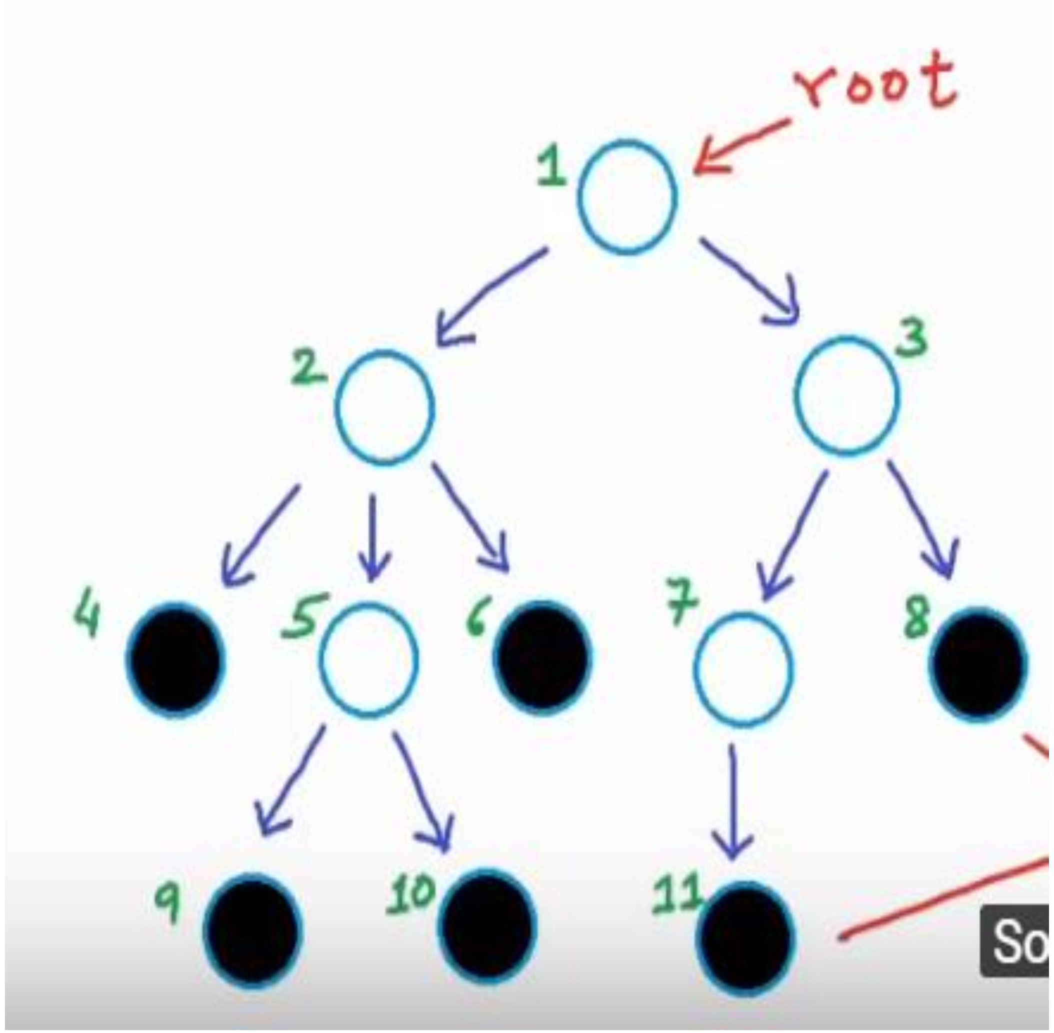
Recursive definition:

- empty tree has no root
- given trees $T_1,\ldots,T_k$ and a node r, there is a tree T where
  - r is the root of T
  - the children of r are the roots of $T_1, \ldots, T_k$



Introduction:





root

children

Parent

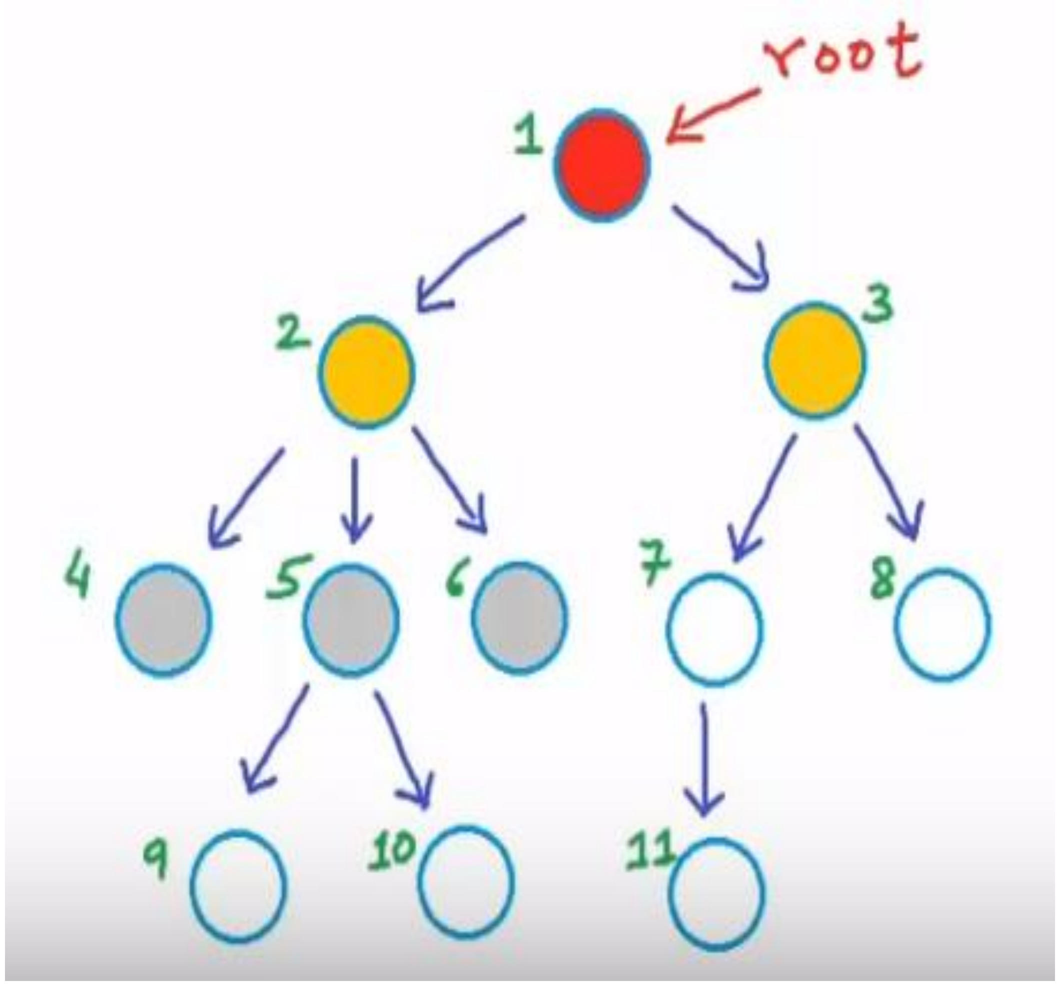
Sibling → have same parent

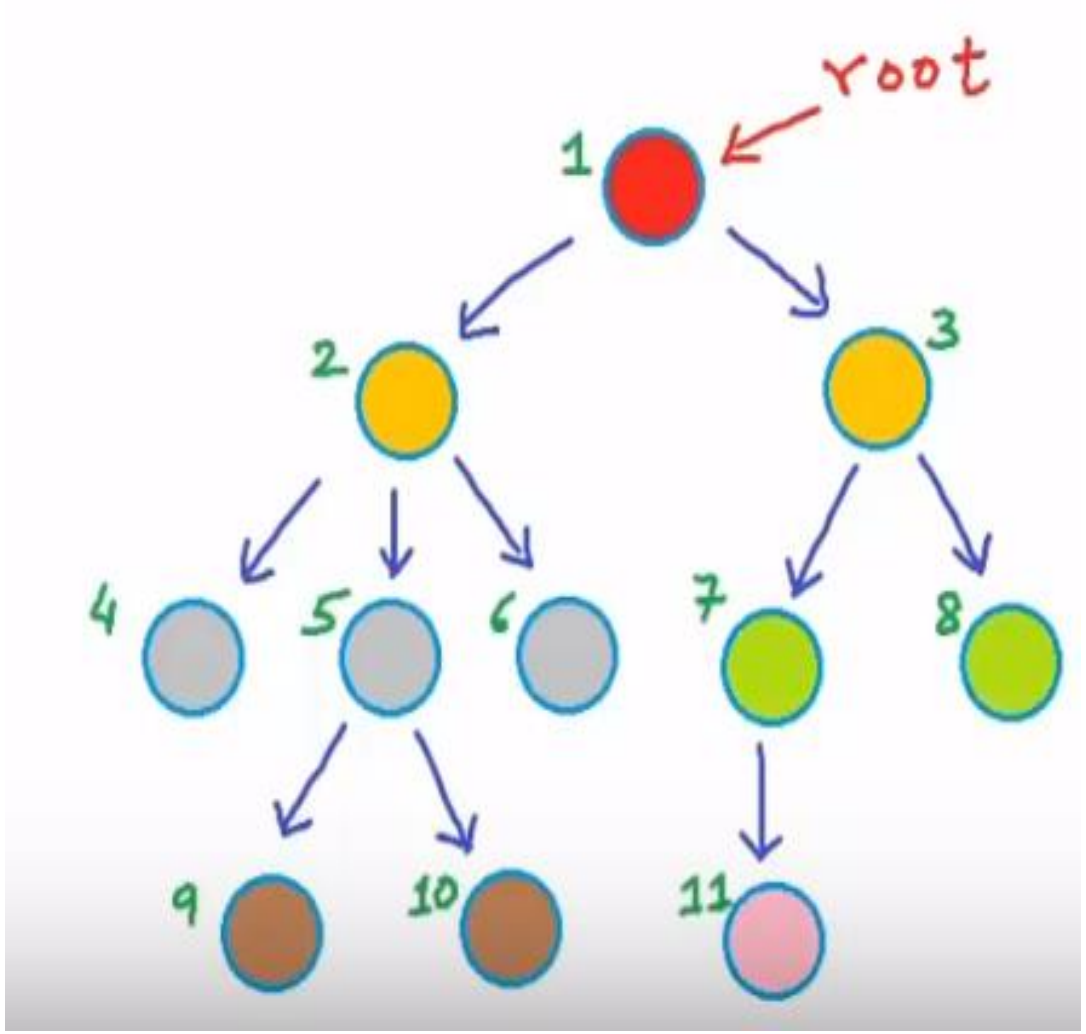
leaf → has no child

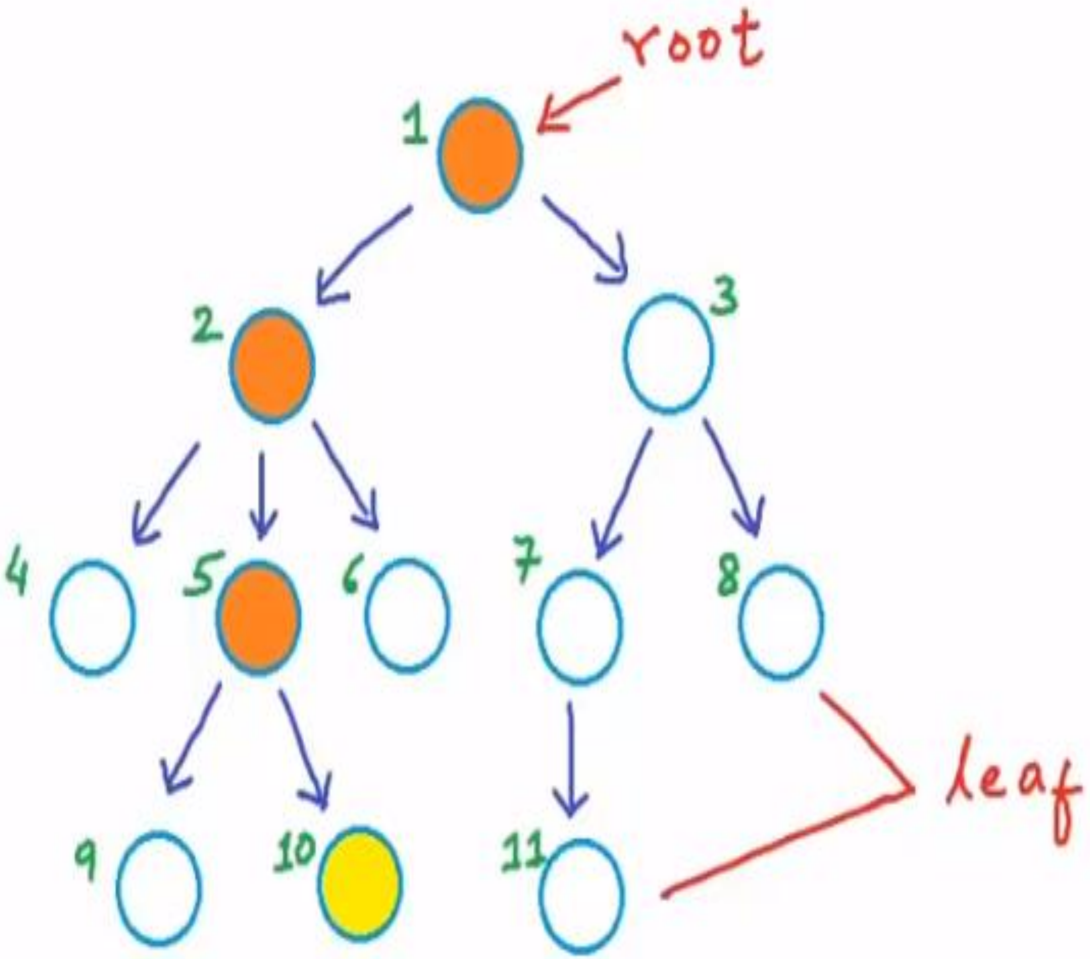
If we can go from A to B

A is ancestor of B

B is descendent of A


root
1
2
3
4
5
6
7
8
9
10
11


root
1
2
3
4
5
6
7
8
9
10
11



root

Children

Parent

Sibling → have same paren

leaf → has no child

If we can go from A to B

A is ancestor of B

B is descendent of A

Depth and Height

Depth of x =

length of path from root to x

OR

No. of edges in path from root to x

Depth and Height

Depth of x =

No. of edges in path from root to x

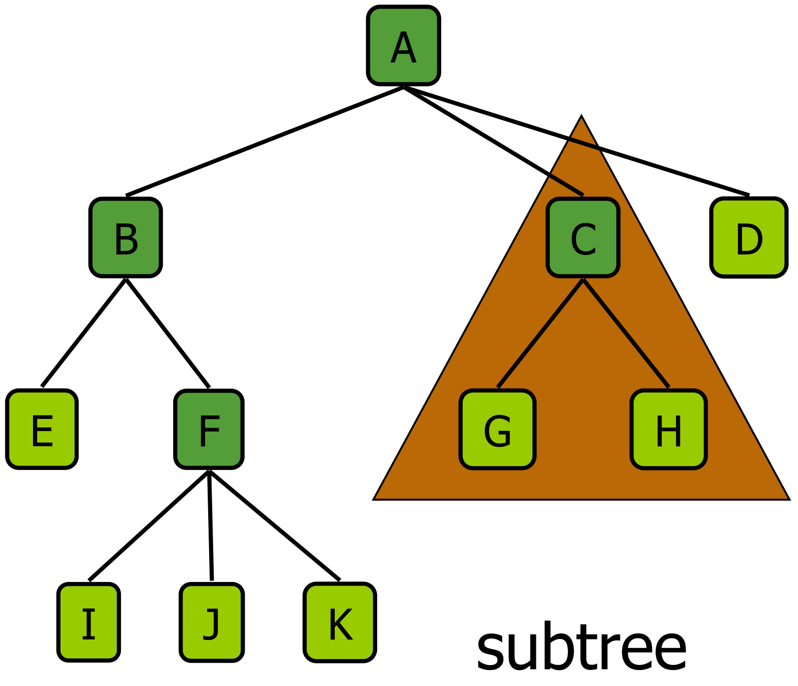
Height of x =

No. of edges in longest path from x to a leaf

Tree Terminology :

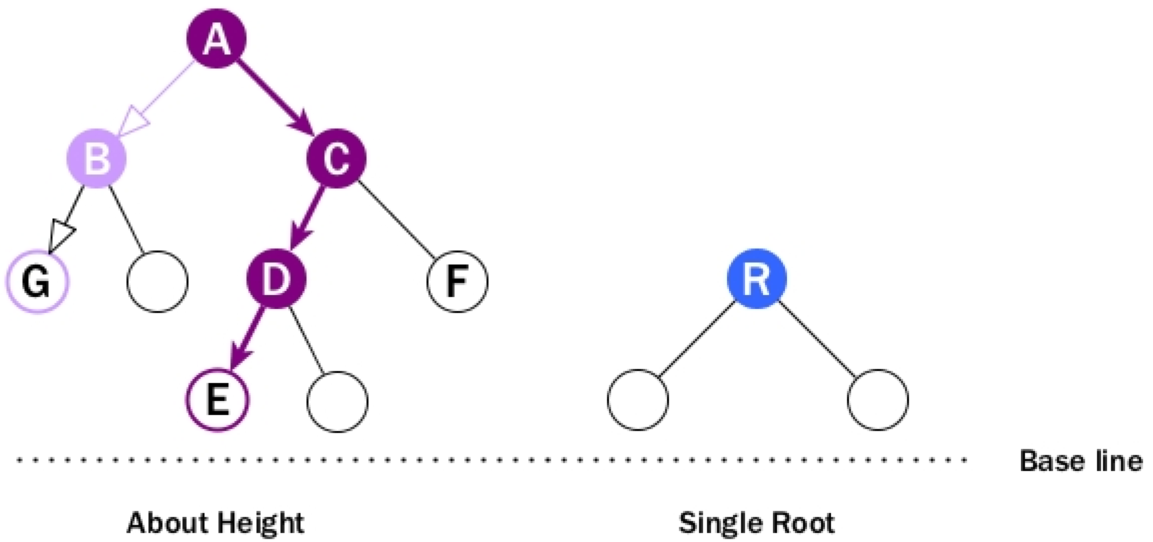
- **Root**: node without parent (A)
- **Siblings**: nodes share the same parent
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node + 1 (4)
- **Degree** of a node: the number of its children

- **Subtree**: tree consisting of a node and its descendants



Height

Height of node – The height of a node is the number of edges on the longest downward path between that node and a leaf.



When looking at height:

1.Every node has height. So *B* can have height, so does *A*, *C* and *D*.

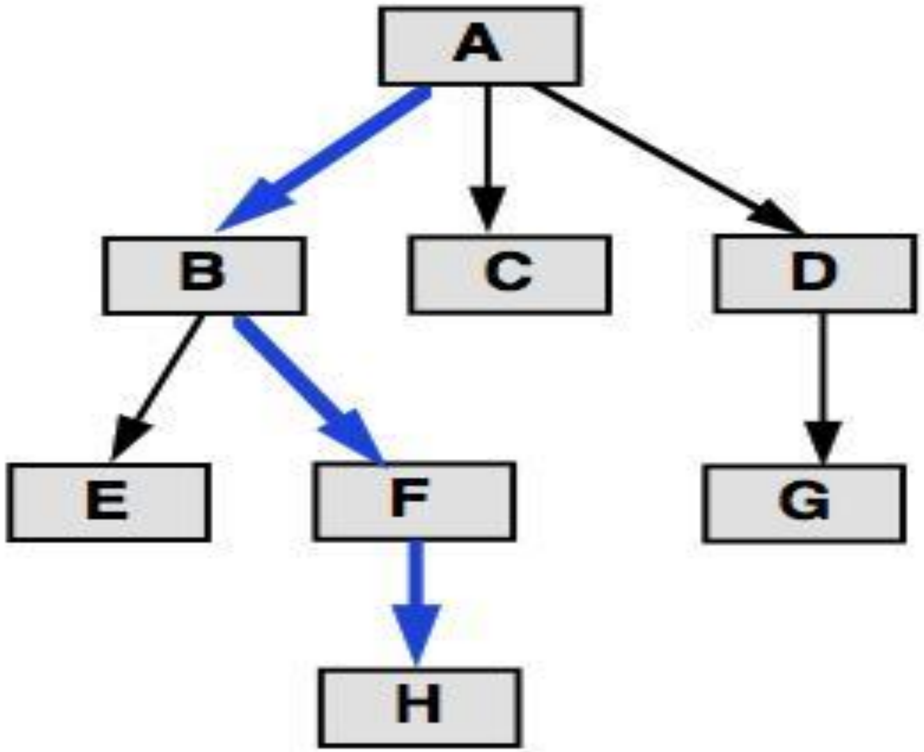
2.Leaf cannot have height as there will be no path starting from a leaf.

3.It is the longest path from the node **to a leaf**. So *A*'s height is the number of edges of the path to *E*, NOT to *G*. And its height is 3.

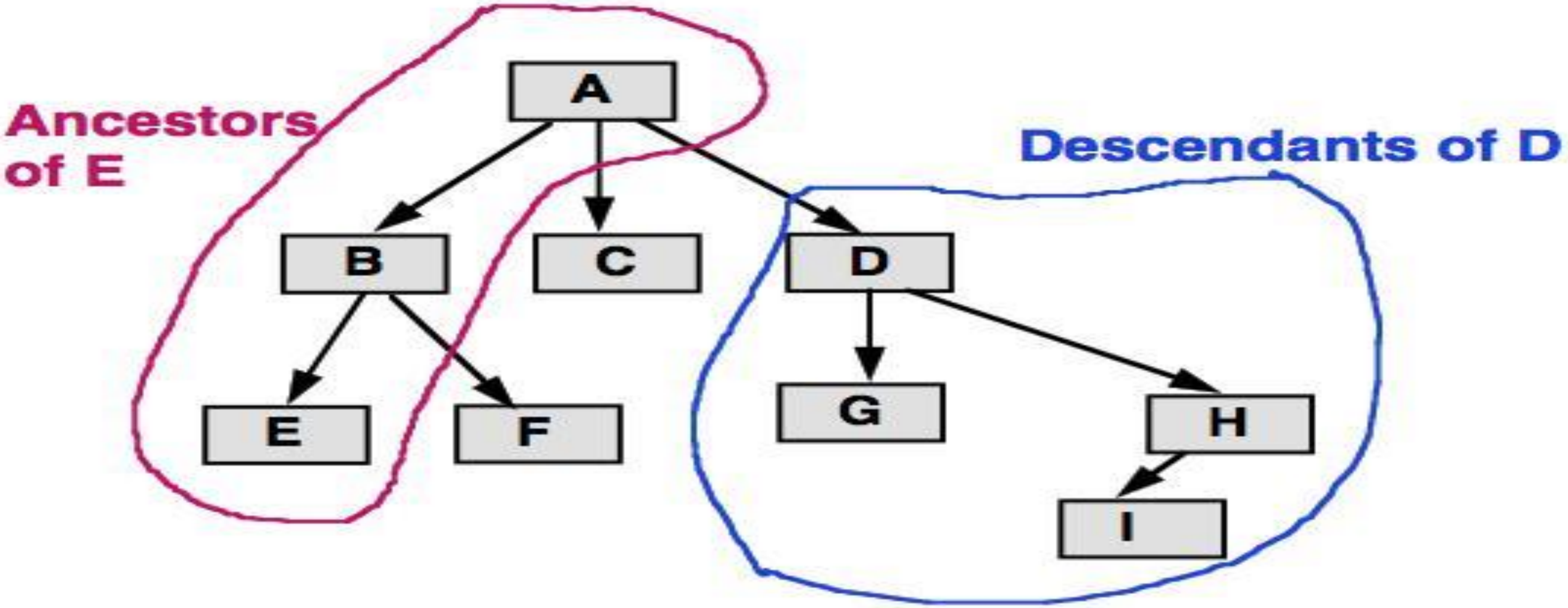
- **Path** – the set of edges from the root to a node
- **Path length** – the number of edges in a path

Path from A to H is
<A,B> <B,F> <F,H>

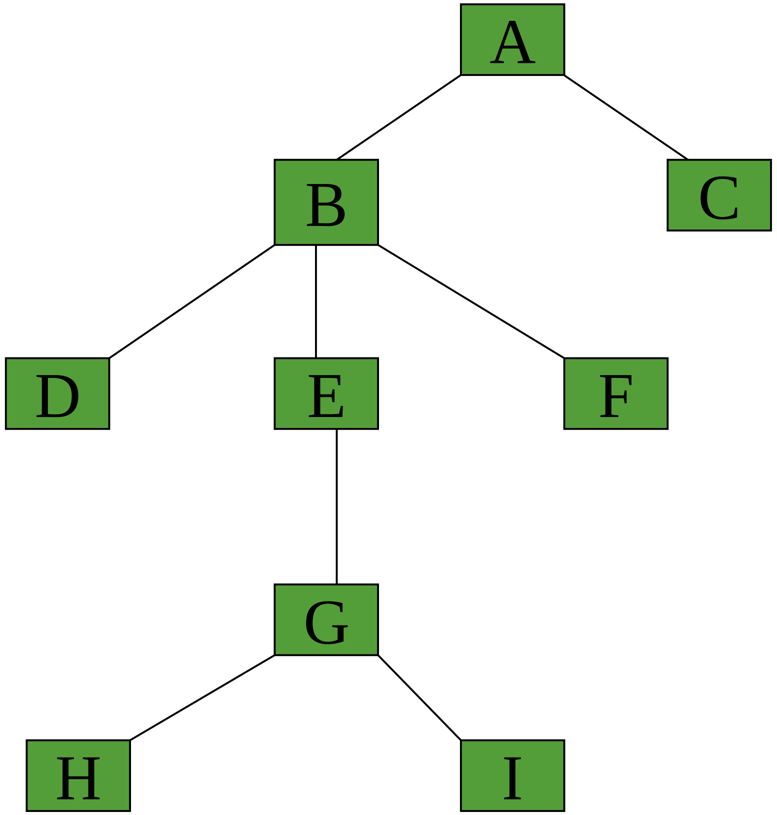
Length is 3



- **Ancestor** – the node itself, parent, parent of parent, etc.
- **Descendent** – the node itself, child, child of child, etc.

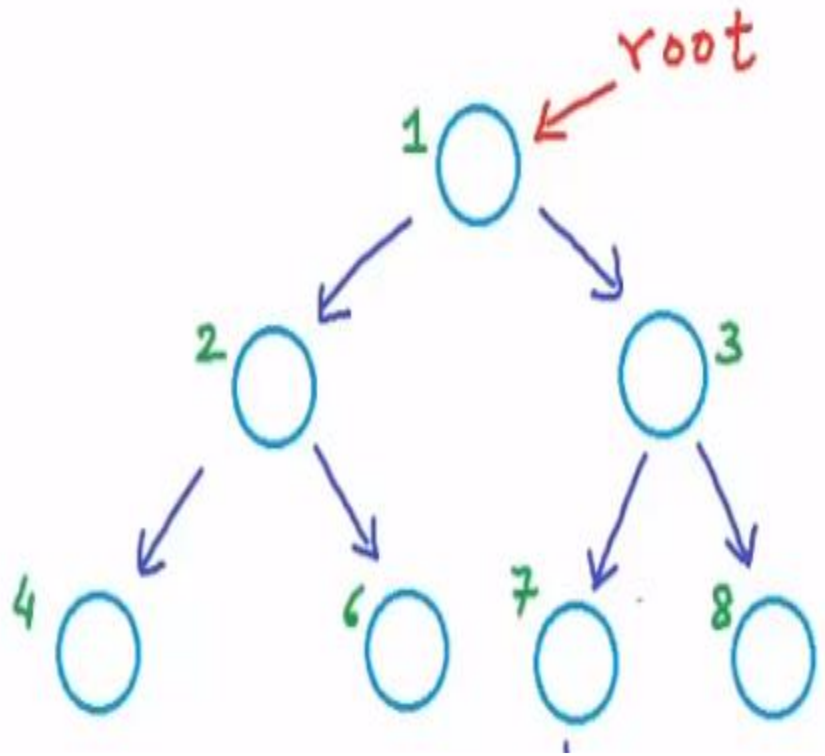


Tree Properties



Property	Value
Number of nodes	
Height	
Root Node	
Leaves	
Interior nodes	
Ancestors of H	
Descendants of B	
Siblings of E	
Right subtree of A	
Degree of B	

Introduction to Binary Tree

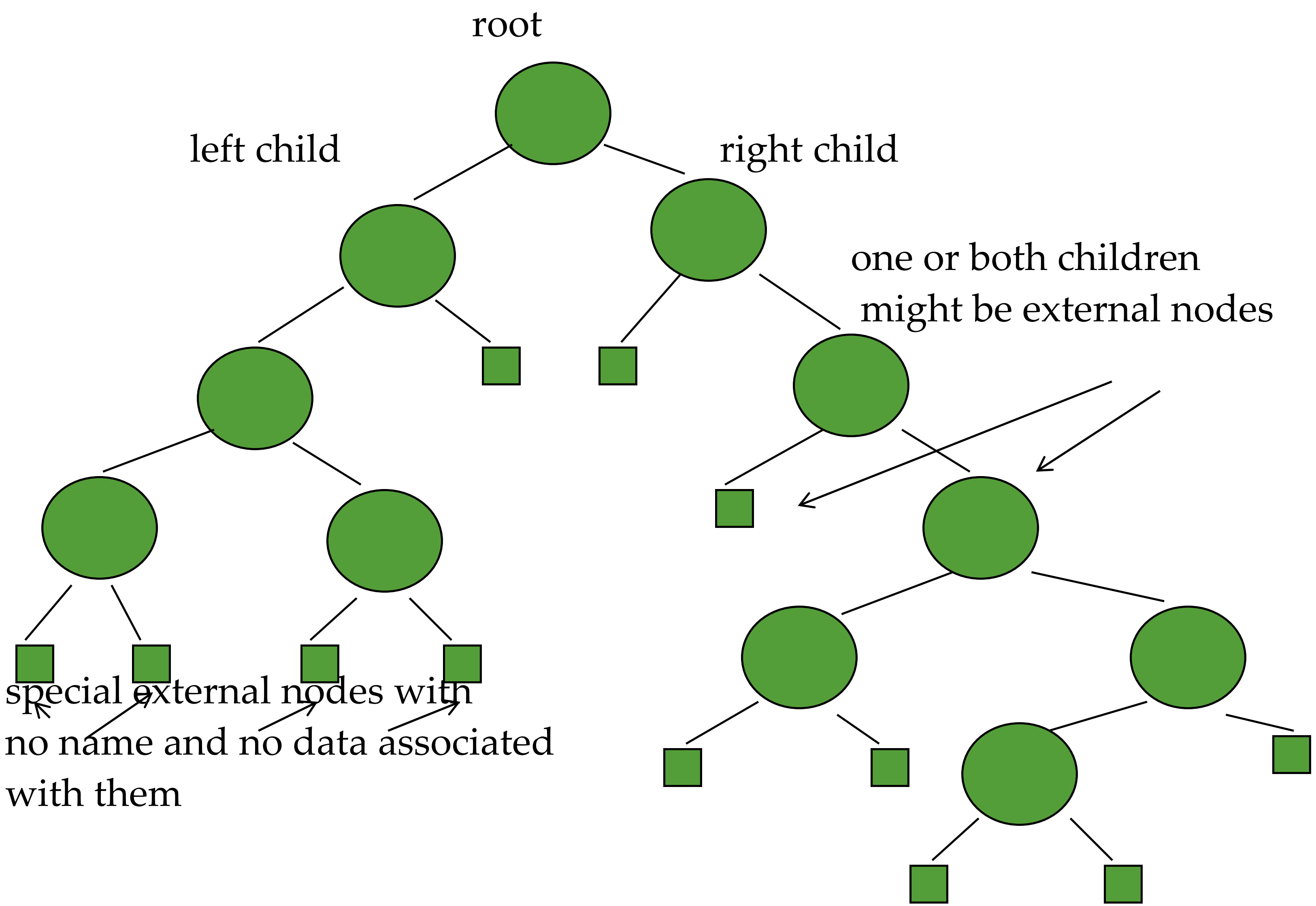


Binary Tree
↓
a tree in which each node can have at most 2 children

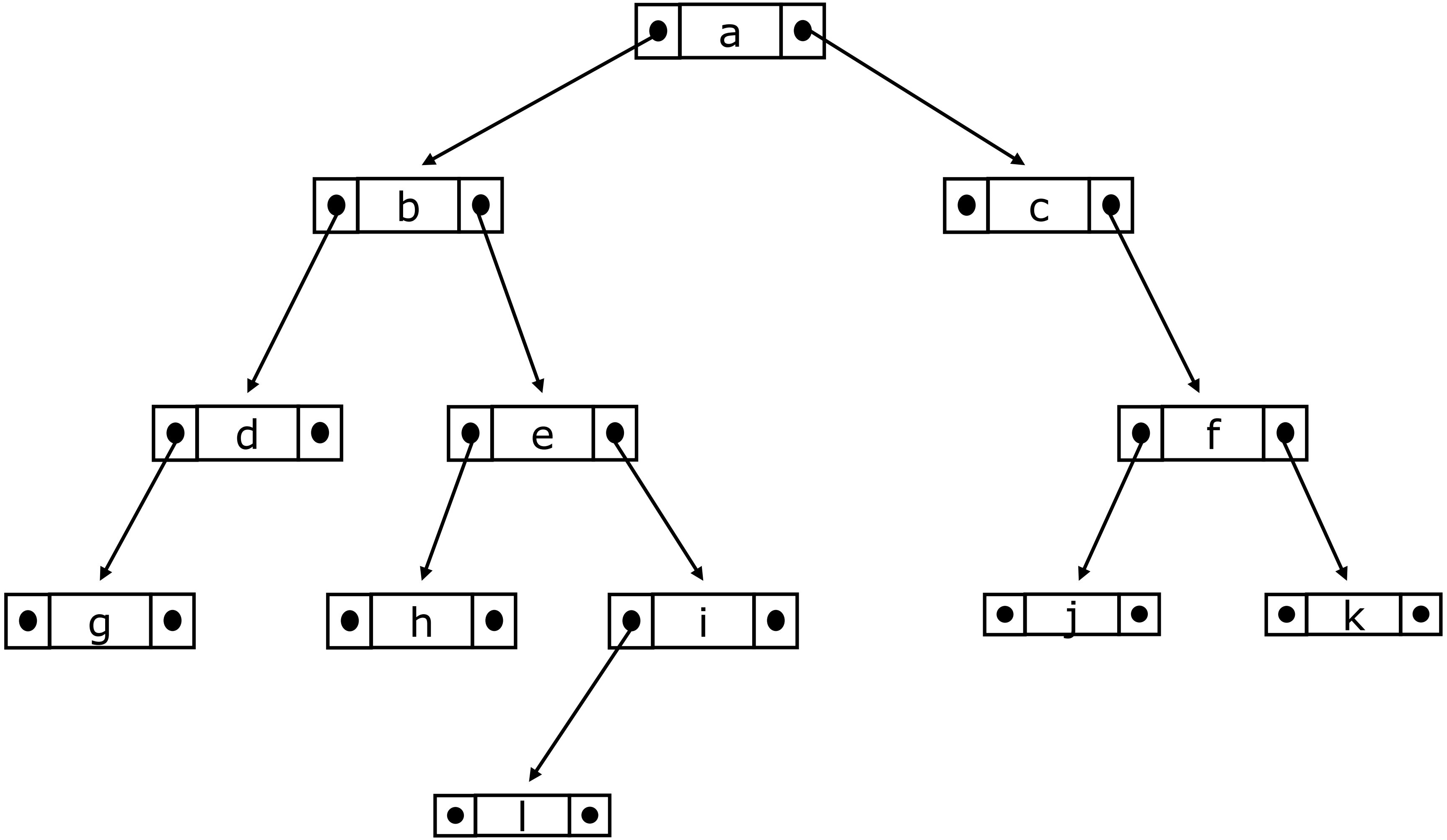
Introduction to Binary Tree:

- A tree where each node has a specific number of children appearing in a specific order is called a **multiway tree**. The simplest type of a multiway tree is the **binary tree**.
  Binary tree is
    - a root
    - left subtree *(maybe empty)*
    - right subtree *(maybe empty)*
- A binary tree is a tree in which no node can have more than two children.
- Each node has an element, a reference to a left child and a reference to a right child.

Example of a binary tree



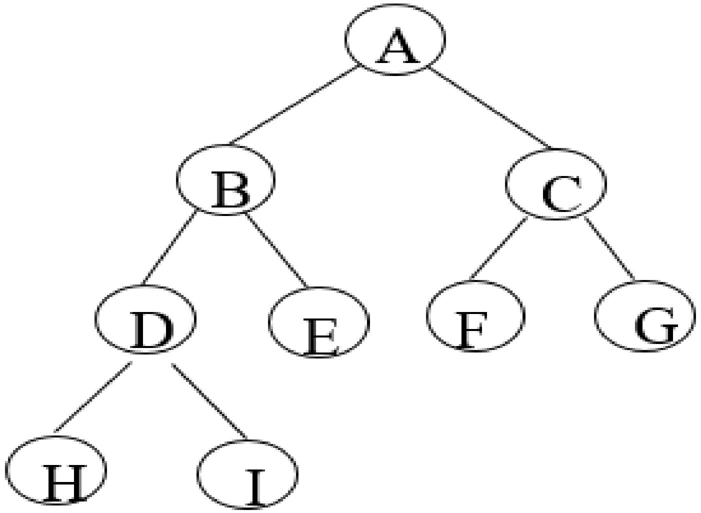
Picture of a binary tree:



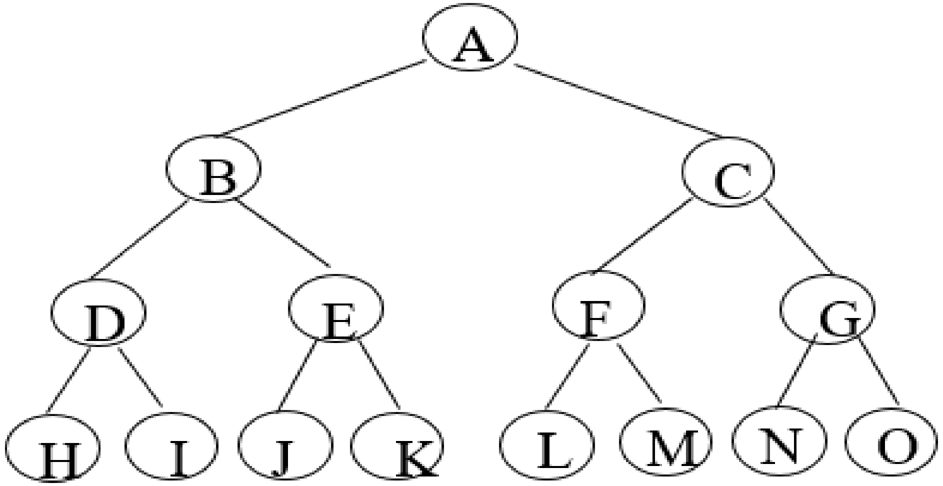
Binary Tree:

- **Binary Tree:** A tree is said to be binary tree if each internal node contain at most 2 children.
- **Strict Binary Tree:** A binary tree is called strict binary tree if each internal node has exactly two child nodes. (Only 0 or 2 children).
- **Full Binary Tree:** Every i^{th} level should have 2^i nodes.
- **Complete Binary Tree:** Except last level every other level should have 2^i nodes and in the last level all the nodes must be as left as possible.

Full BT and Complete BT:



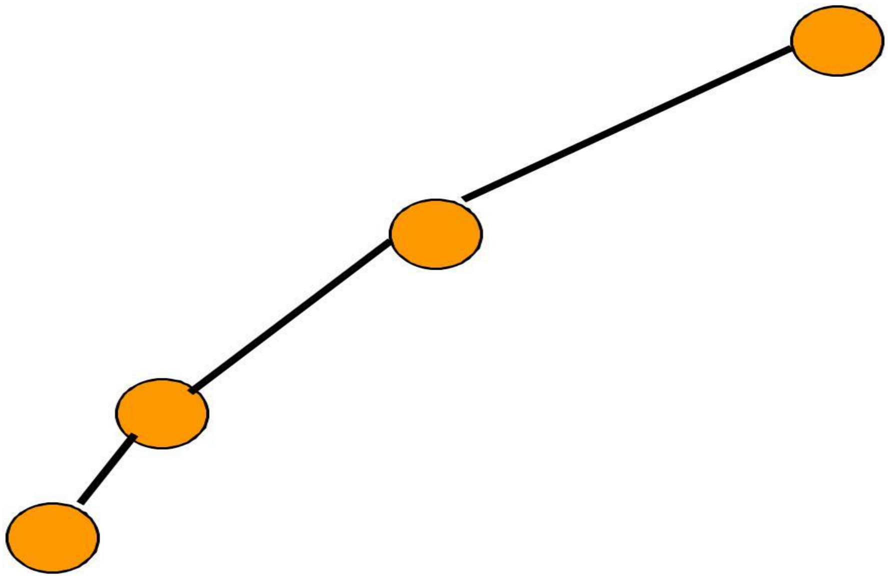
s Complete binary tree



Full binary tree of depth 4

Minimum Number Of Nodes

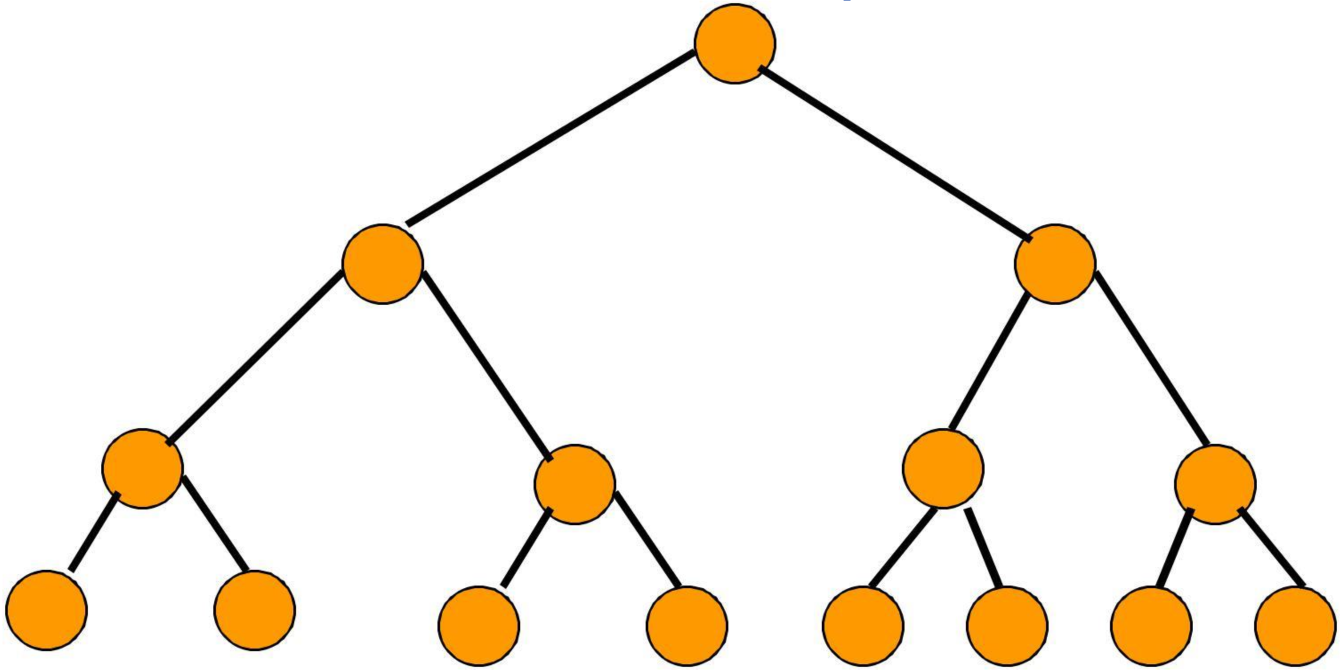
- Minimum number of nodes in a binary tree whose height is h.
- At least one node at each of first h levels.



minimum number of nodes is h

Maximum Number Of Nodes:

- All possible nodes at first h levels are present.



Maximum number of nodes

$= 1 + 2 + 4 + 8 + \ldots + 2^h$

$= 2^{h+1} - 1$

Number Of Nodes & Height

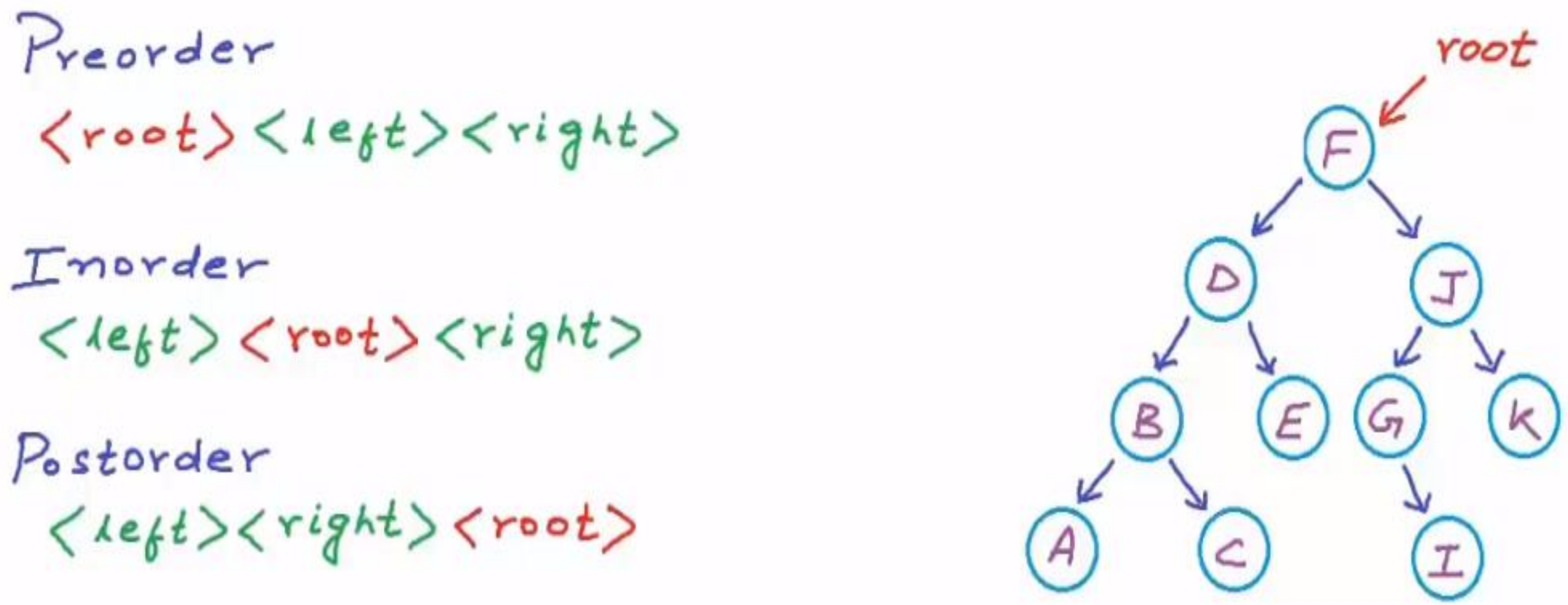
Let n be the number of nodes in a complete binary tree whose height is h.

- $2^h <= n <= 2^{h+1} - 1$

Tree Traversals:

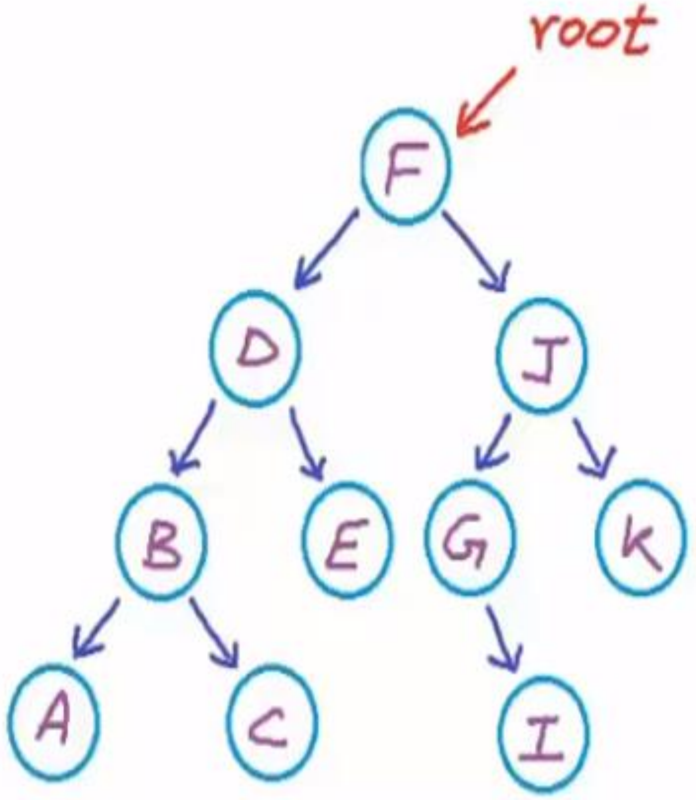
- A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree
- To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once
- Tree traversals are naturally recursive
- Since a binary tree has three "parts," there are three possible ways to traverse the binary tree:
  - root, left, right
  - left, root, right
  - left, right, root

Tree Traversal:



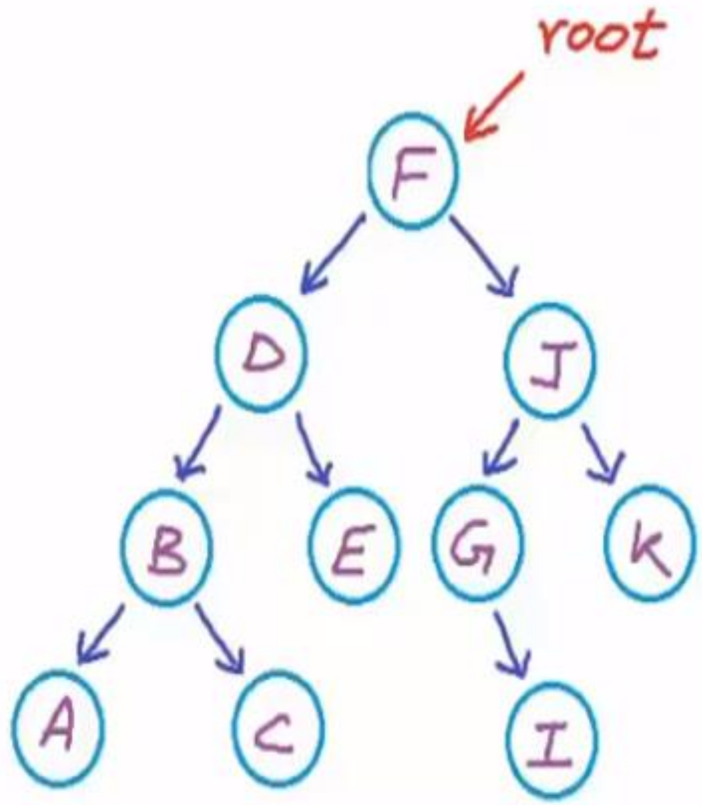
Preorder Traversal :

```
void preOrder(tree_node ptr)
{
    if(ptr != Null)
    {
        visit(t);
        preOrder(ptr.leftChild);
        preOrder(ptr.rightChild);
    }
}
```



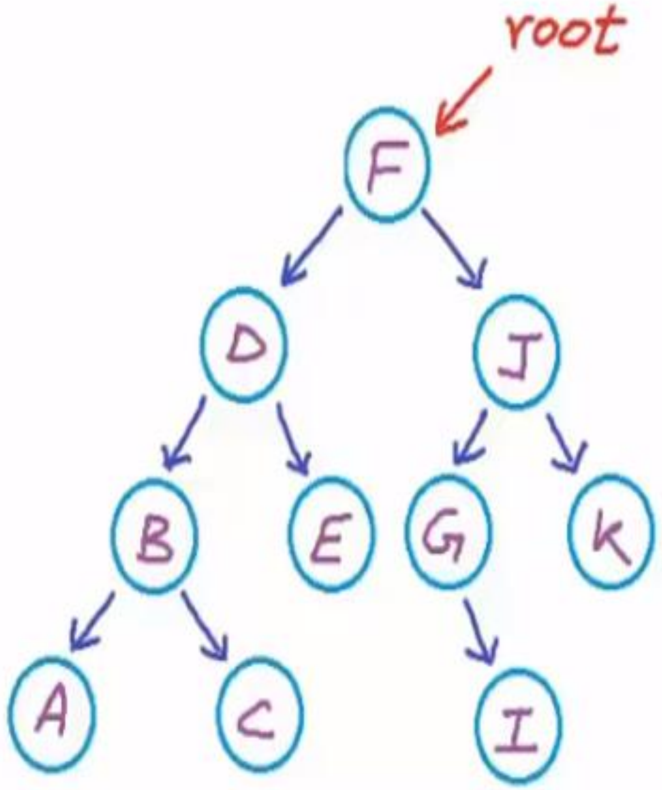
Inorder Traversal:

```
void inOrder(tree_node ptr)
{
   if(ptr != Null)
    {
        inOrder(ptr.leftChild);
        visit(ptr);
        inOrder(ptr.rightChild);
    }
}
```



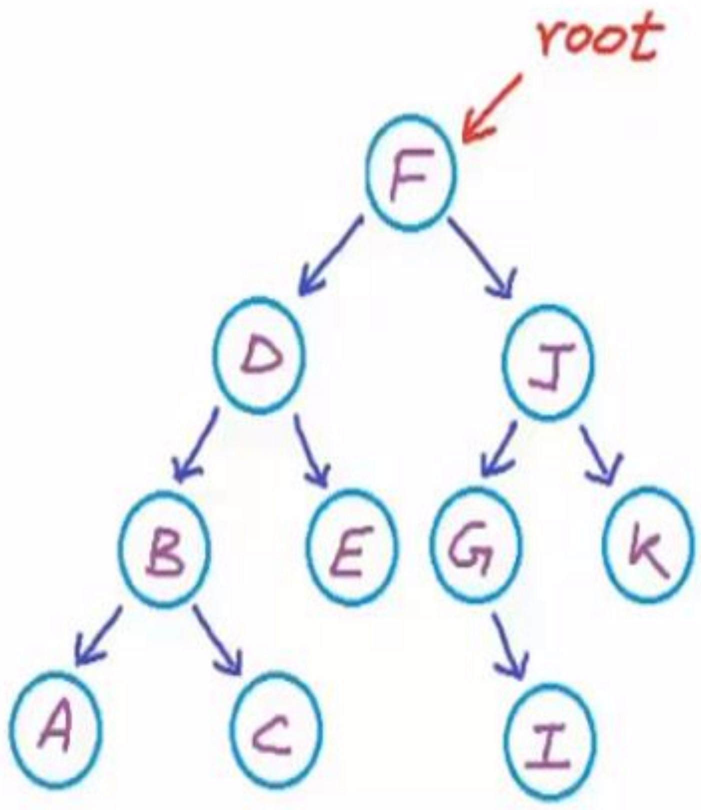
Postorder Traversal:

```
void postOrder(tree_node ptr)
{
   if(ptr != Null)
    {
        postOrder(ptr.leftChild);
        postOrder(ptr.rightChild);
        visit(t);
    }
}
```

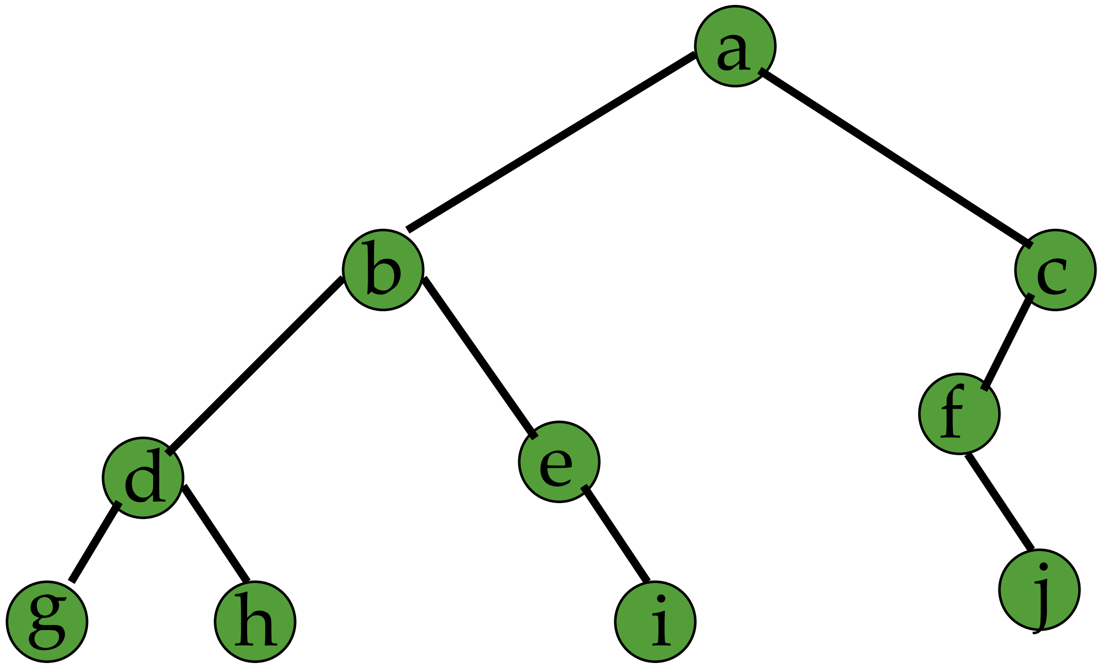


Tree Traversal:

- Preorder: **F D B A C E J G I K**
- Inorder : **A B C D E F G I J K**
- PostOrder: **A C B E D I G K J F**



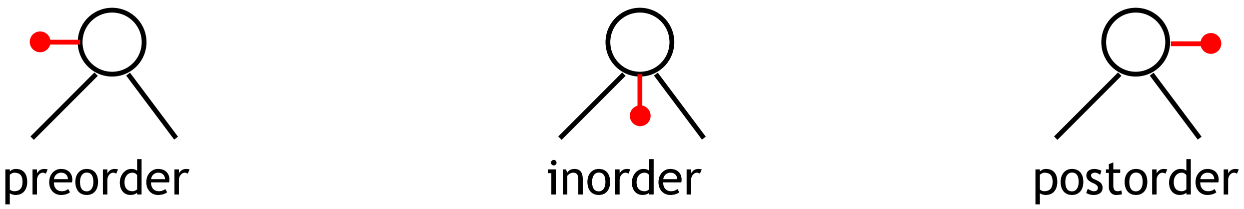
Level-Order Example (Visit = print)



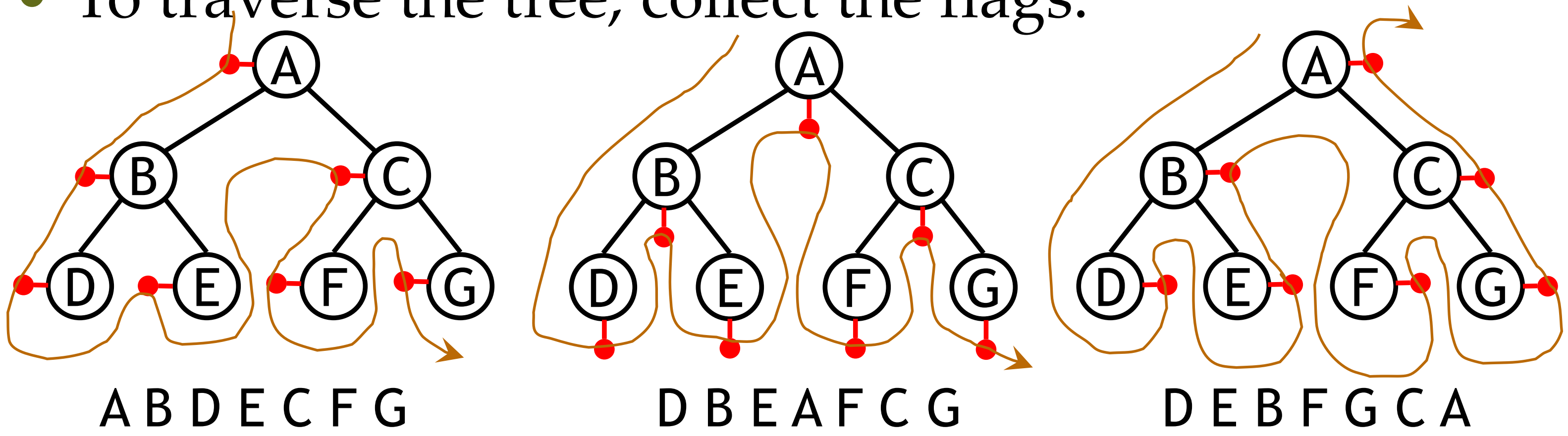
a b c d e f g h i j

Tree traversals using "flags"

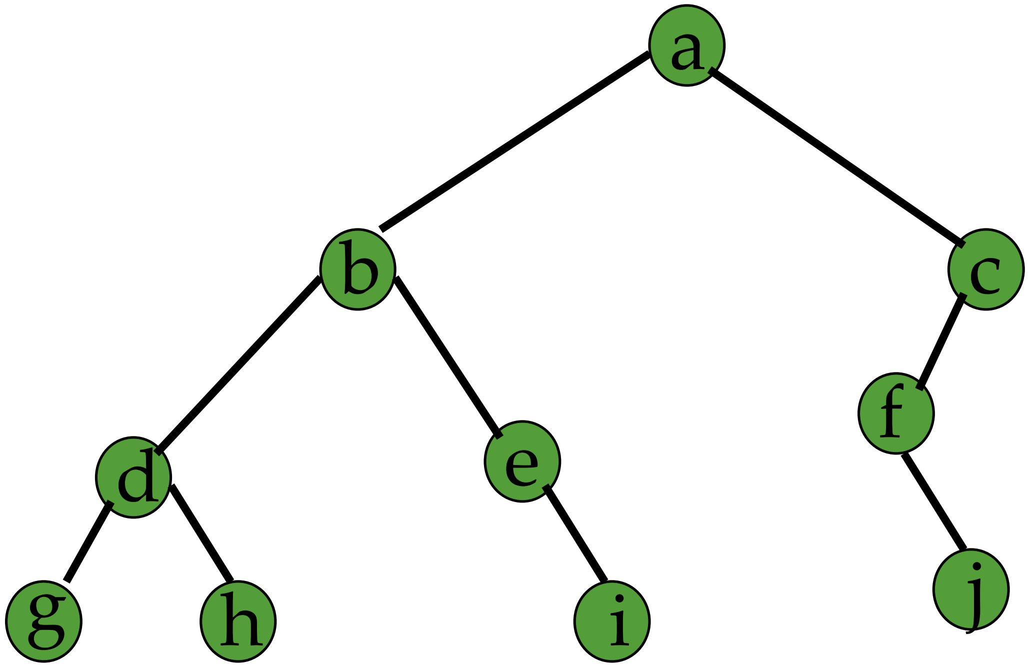
- The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a "flag" attached to each node, as follows:



- To traverse the tree, collect the flags:



Tree Traversal Practice:

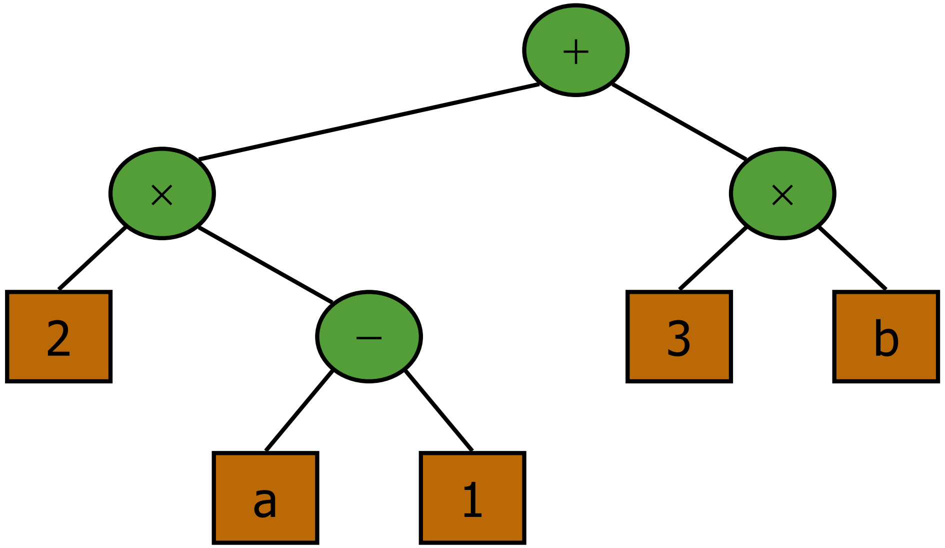


Tree Traversal:

- Preorder: **a b d g h e i c f j**
- Inorder : **g d h b e i a f j c**
- PostOrder: **g h d i e b j f c a**

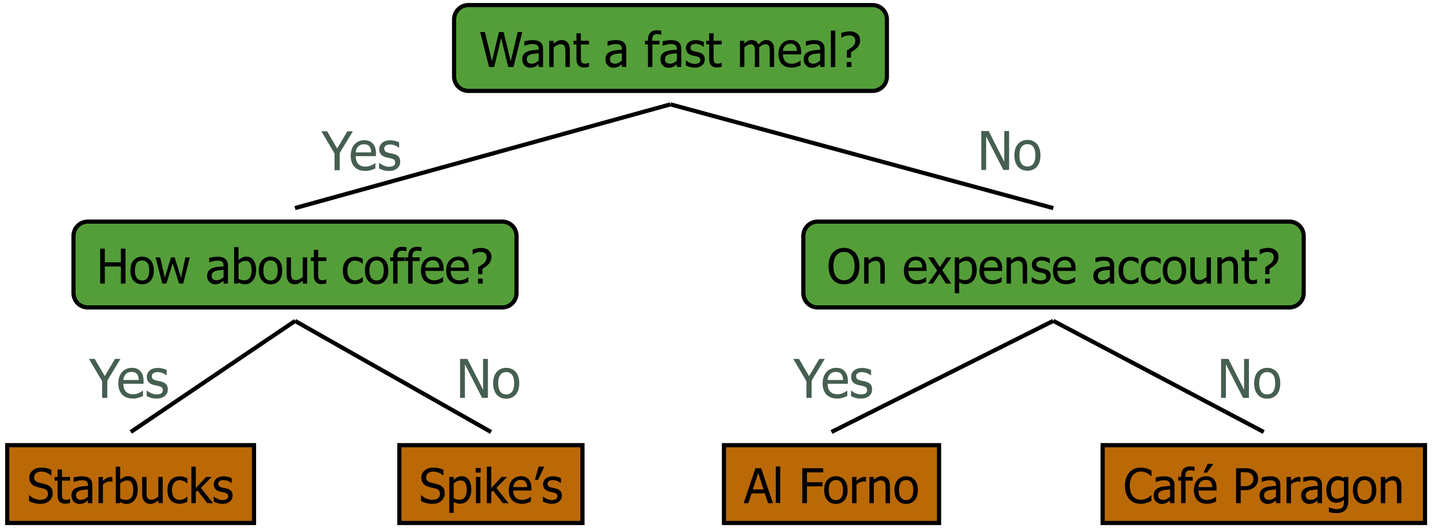
Arithmetic Expression Tree:

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



Decision Tree :

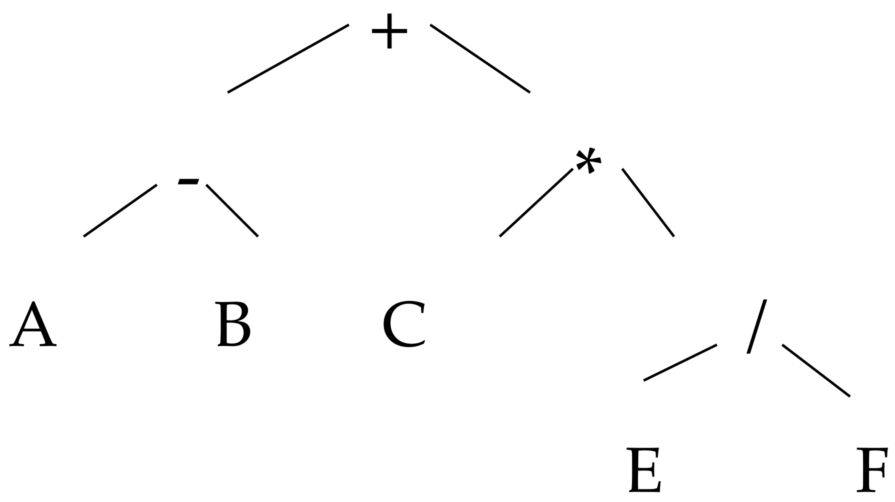
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



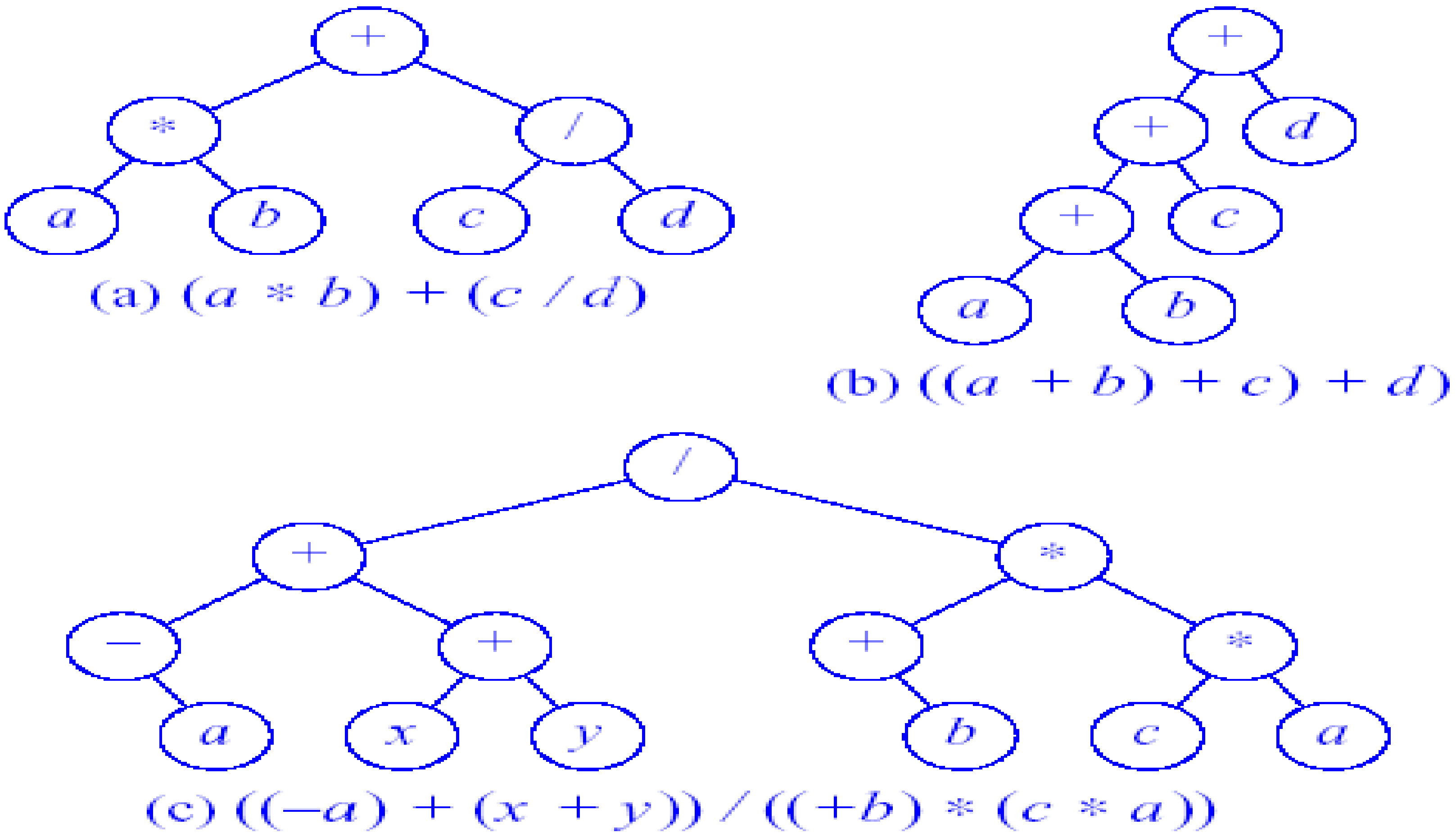
More binary trees examples

- Binary tree for representing arithmetic expressions. The underlying hierarchical relationship is that of an arithmetic operator and its two operands.

Arithmetic expression in an infix form: (A - B) + C * (E / F)



Binary Tree for Expressions



Expression Trees

Example: Expression Trees

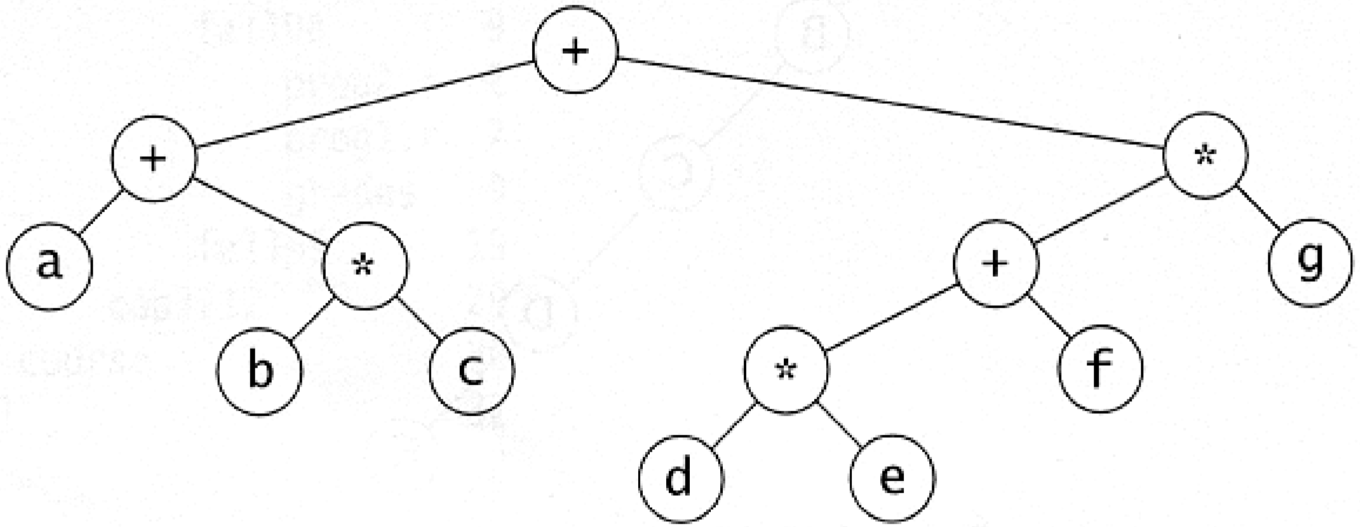
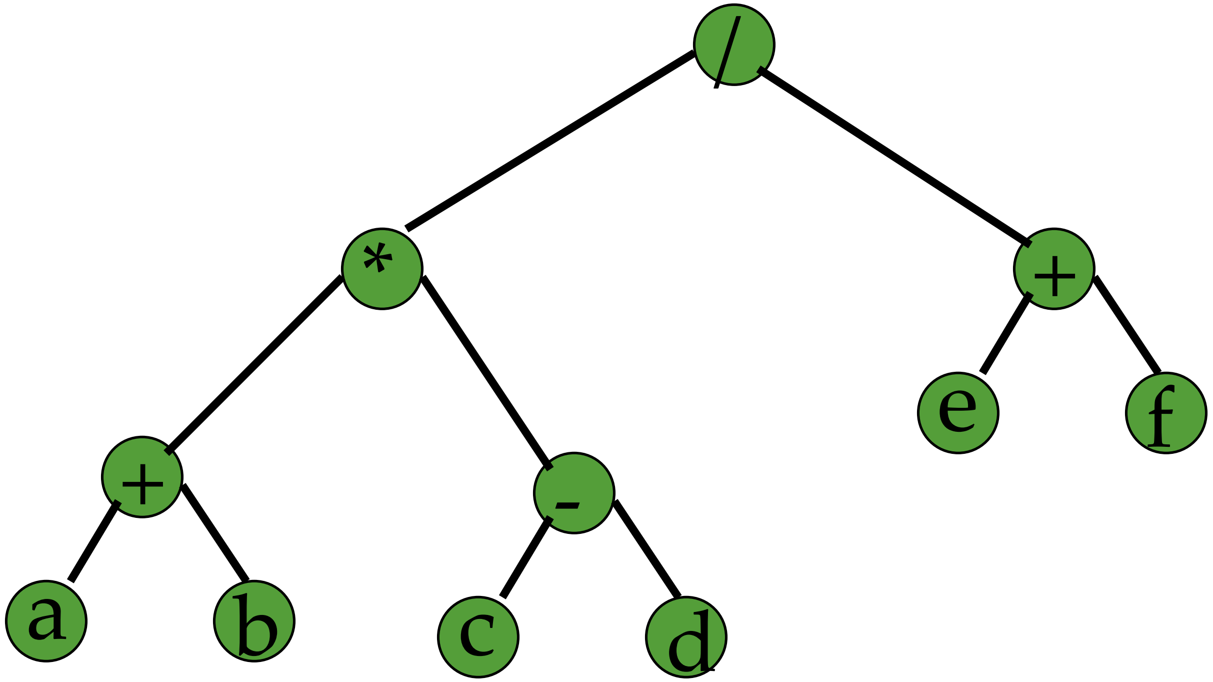


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

- Leaves are operands (constants or variables)
- The other nodes (internal nodes) contain operators

Expression Tree Traversal:



Expression Tree Traversal:

- Preorder : **/ * + a b – c d + e f** Gives prefix form of expression!
- Inorder : **a + b * c – d / e + f** Gives infix form of expression!
- Postorder : **a b + c d - * e f + /** Gives postfix form of expression!

Binary Tree Construction

- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given.

Preorder And Postorder

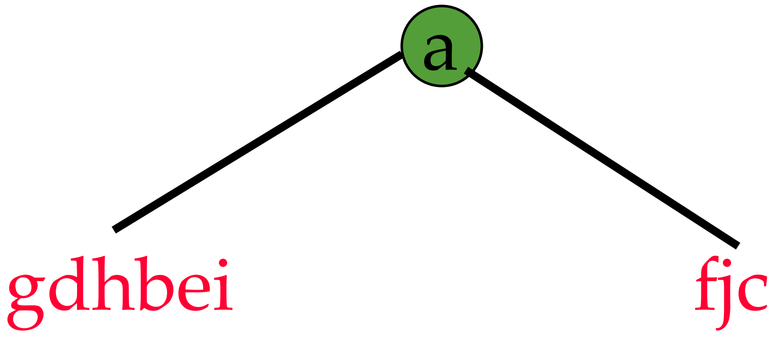
preorder = ab

postorder = ba

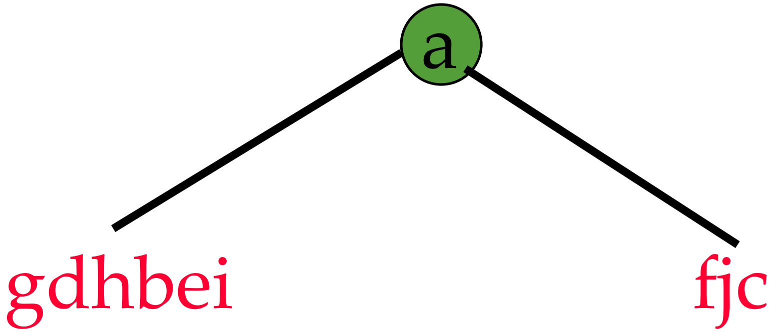
- Preorder and postorder do not uniquely define a binary tree.
- Nor do preorder and level order (same example).
- Nor do postorder and level order (same example).

Inorder And Preorder

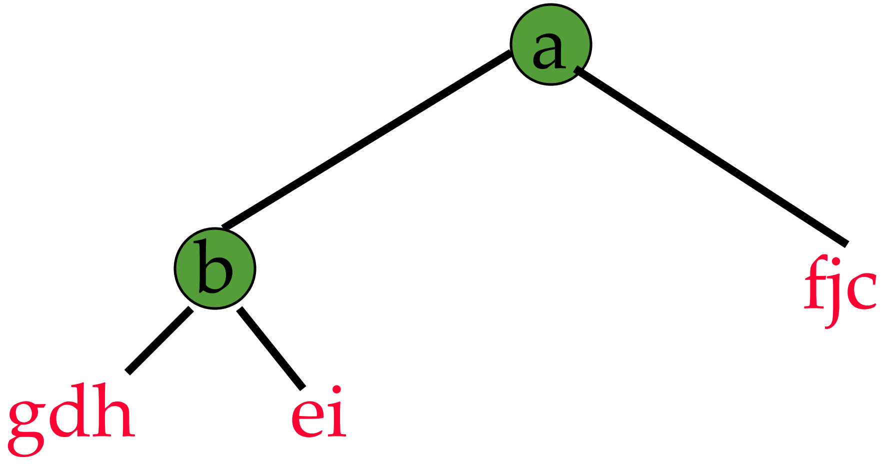
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees.
- a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree.



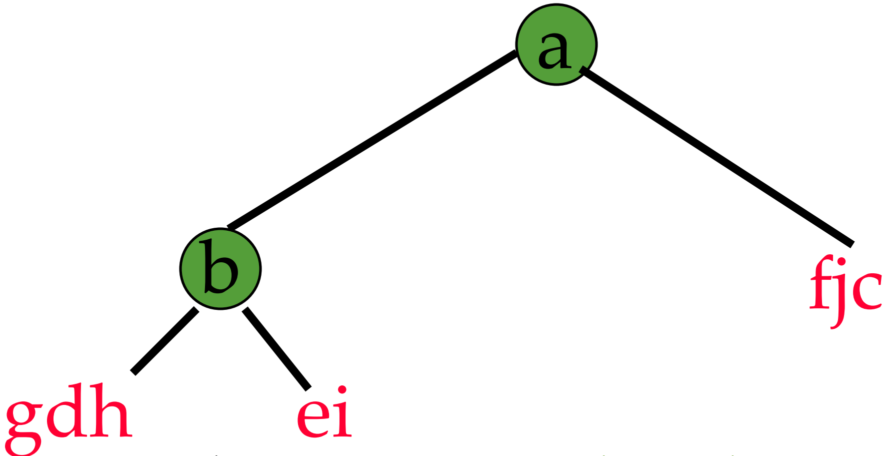
Inorder And Preorder



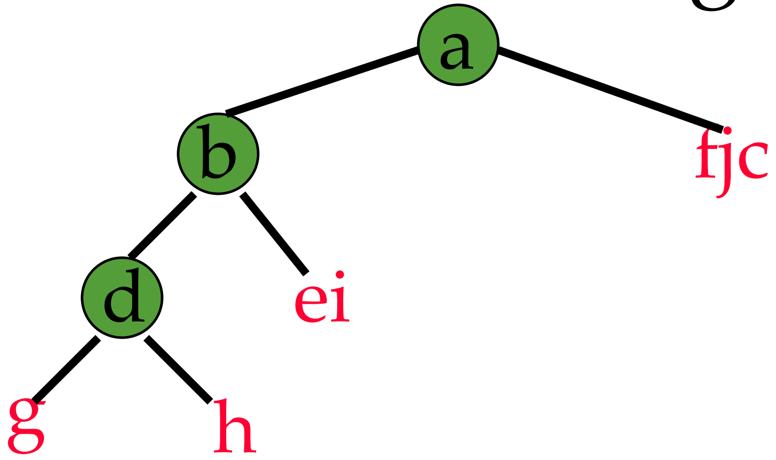
- preorder = b d g h e i c f j
- b is the next root; gdh are in the left subtree; ei are in the right subtree.



Inorder And Preorder



- preorder = d g h e i c f j
- d is the next root; g is in the left subtree; h is in the right subtree.



Inorder And Postorder

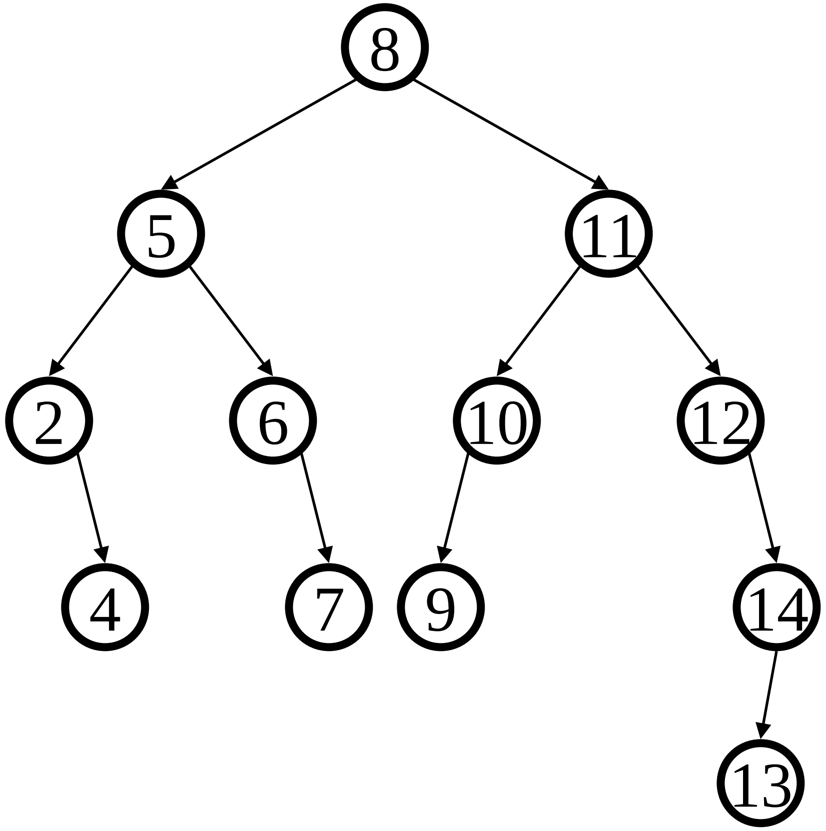
- Scan postorder from right to left using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Inorder And Level Order

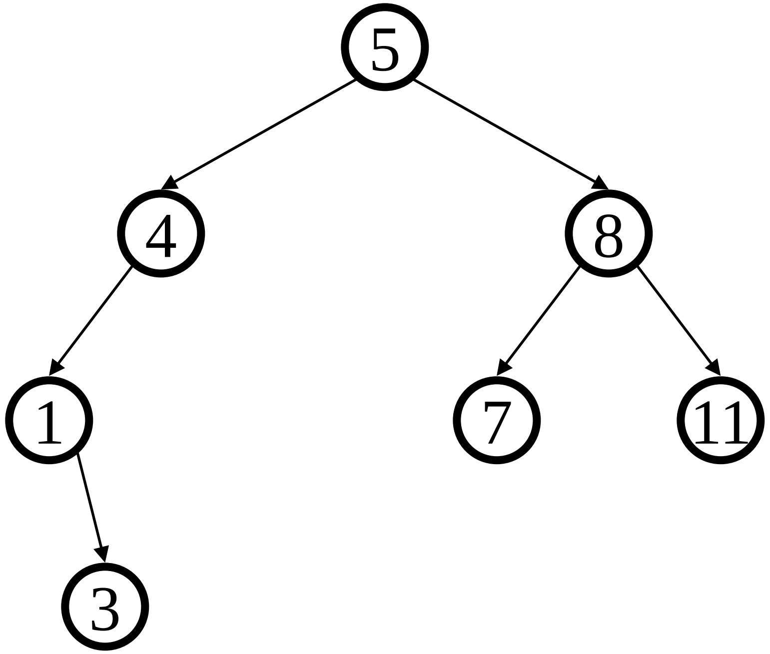
- Scan level order from left to right using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- level order = a b c d e f g h i j
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Binary Search Tree :

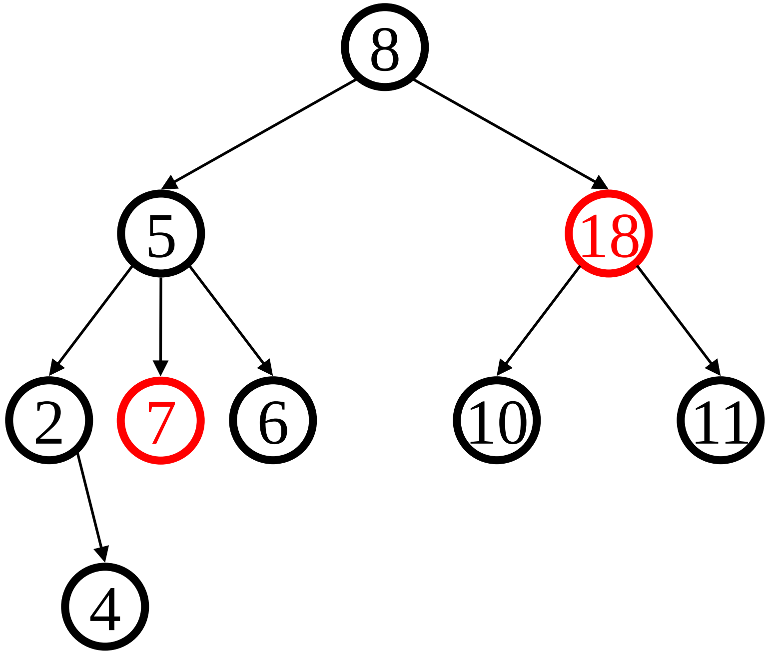
- Search tree property
  - all keys in left subtree smaller than root's key
  - all keys in right subtree larger than root's key
  - result:
    - easy to find any given key
    - inserts/deletes by changing links



Example and Counter-Example



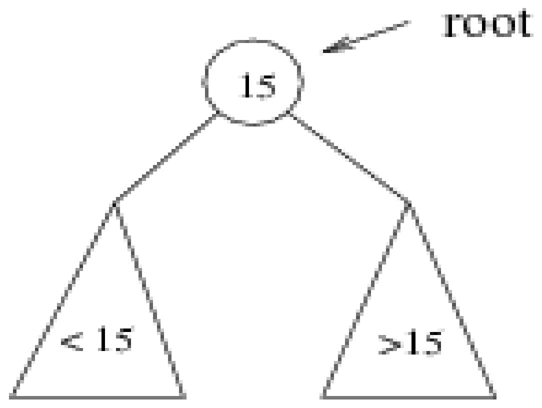
BINARY SEARCH TREE



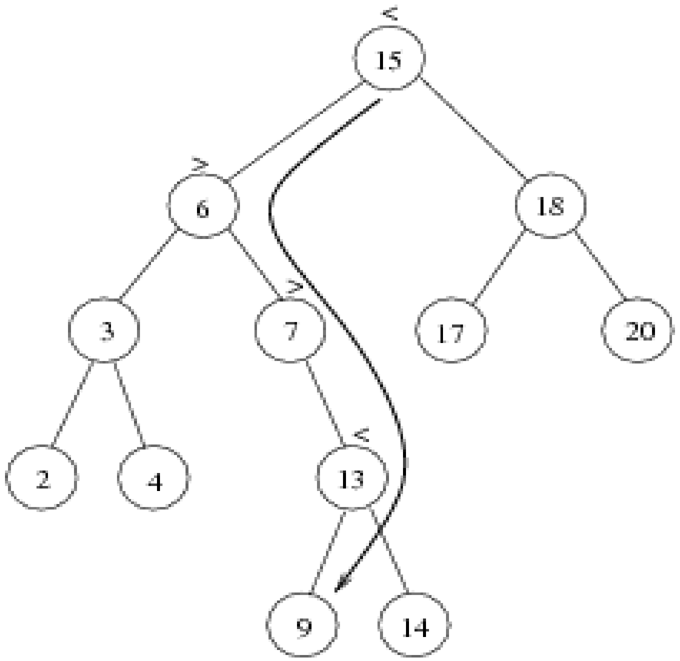
NOT A
BINARY SEARCH TREE

Searching BST:

- If we are searching for 15, then we are done.
- If we are searching for a key < 15, then we should search in the left subtree.
- If we are searching for a key > 15, then we should search in the right subtree.



Example: Search for 9 ...



Search for 9:

1. compare 9:15(the root), go to left subtree;
2. compare 9:6, go to right subtree;
3. compare 9:7, go to right subtree;
4. compare 9:13, go to left subtree;
5. compare 9:9, found it!