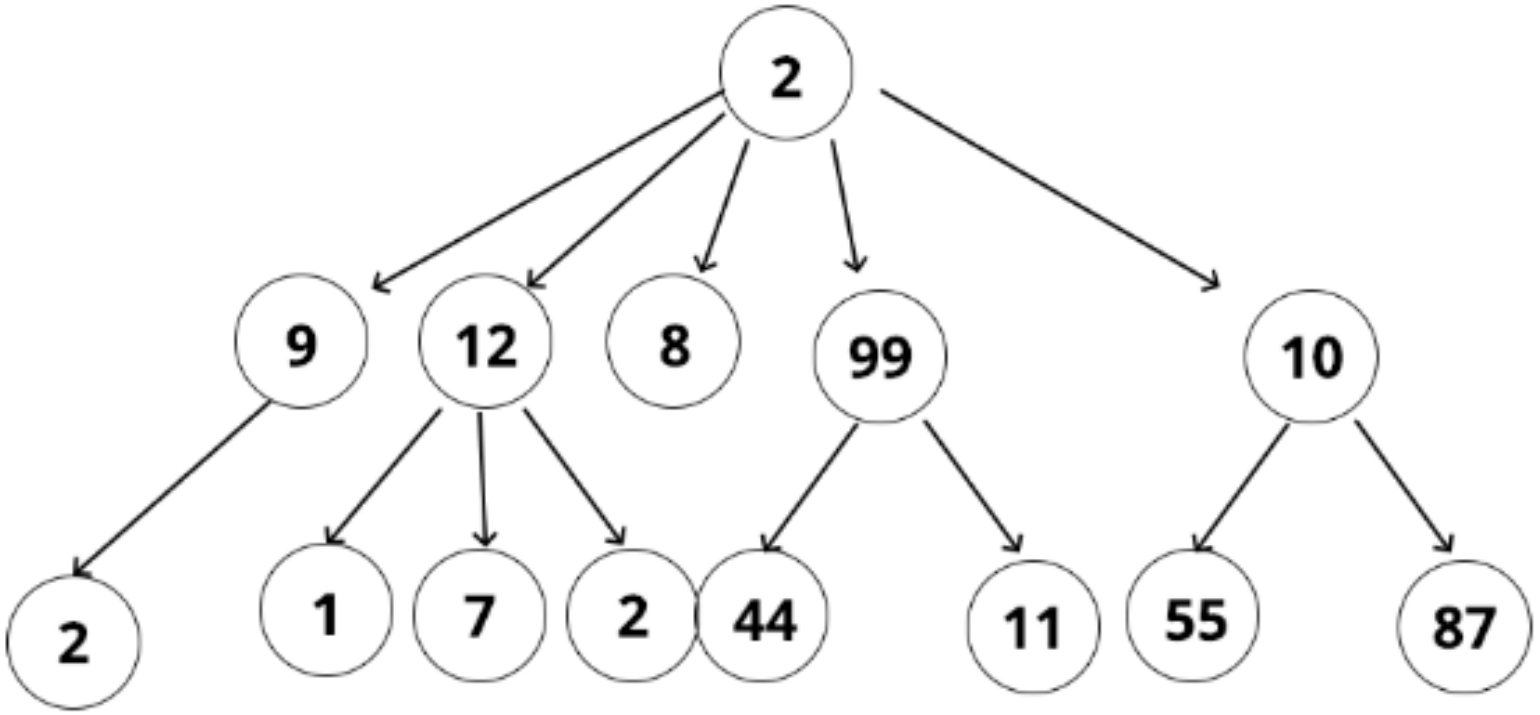
Trees

TREES



What are trees?

Tree is a hierarchical data structure which stores the information naturally in the form of hierarchy style.

Tree is one of the most powerful and advanced data structures.

It is a non-linear data structure compared to arrays, linked lists, stack and queue.

It represents the nodes connected by edges.

Definition of Tree

- A tree *t* is a finite nonempty set of elements
- One of these elements is called the root
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of *t*.

Why trees ?

- Describe dynamic properties of algorithms
- Build and use explicit data structures that are concrete realizations of trees

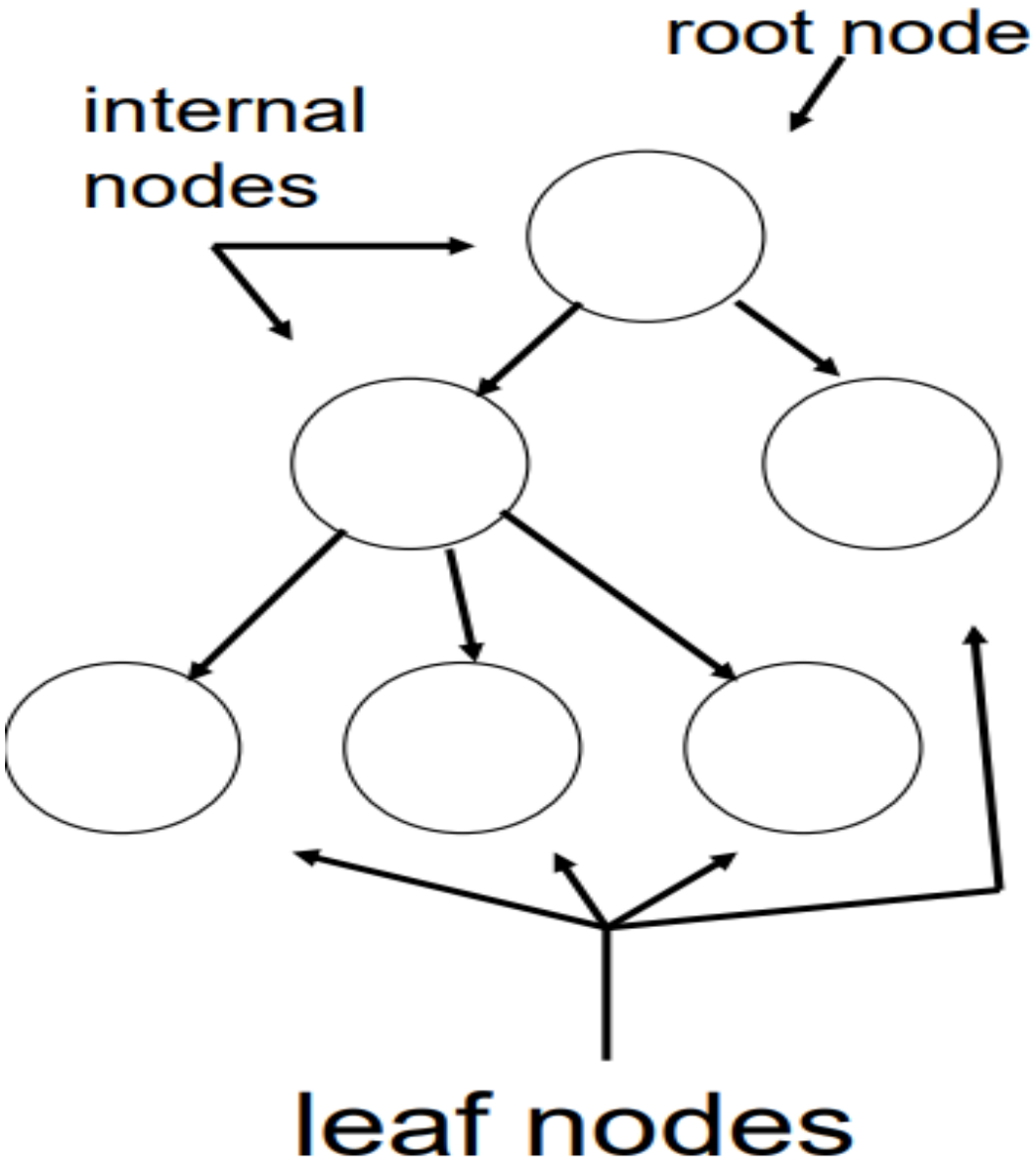
A tree is a data structure accessed beginning at a root node

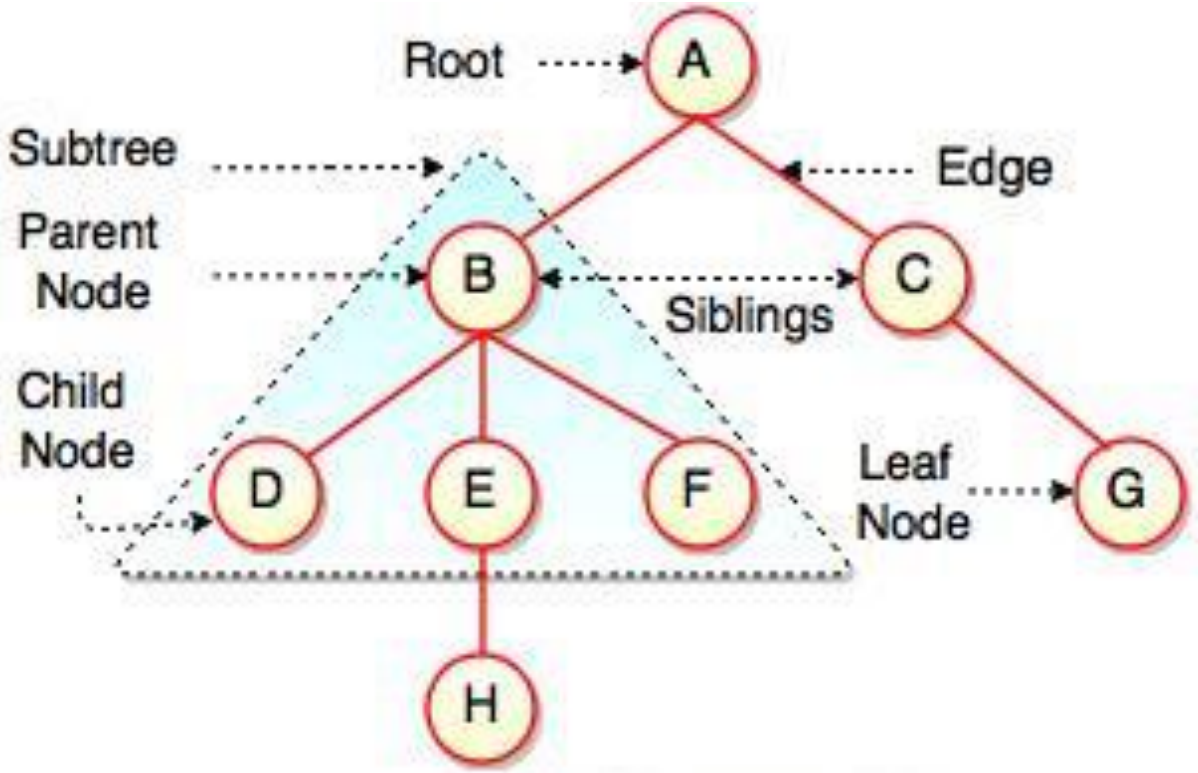
- Each node is either a leaf or an internal node
- An internal node has 1 or more children, nodes that can be reached directly from that internal node.
- The internal node is said to be the parent of its child nodes

Definition (recursively):

A tree is a finite set of one or more nodes such that

- There is a specially designated node called **root**.
- The remaining nodes are partitioned into n>=0 disjoint set $T_1,\ldots,T_n$, where each of these sets is a tree. $T_1,\ldots,T_n$ are called the **subtrees** of the root.
- Every node in the tree is the **root** of some **subtree**


root node
internal
nodes
leaf nodes



The above figure represents structure of a tree. Tree has 2 subtrees.

A is a parent of B and C. B is called a child of A and also parent of D,

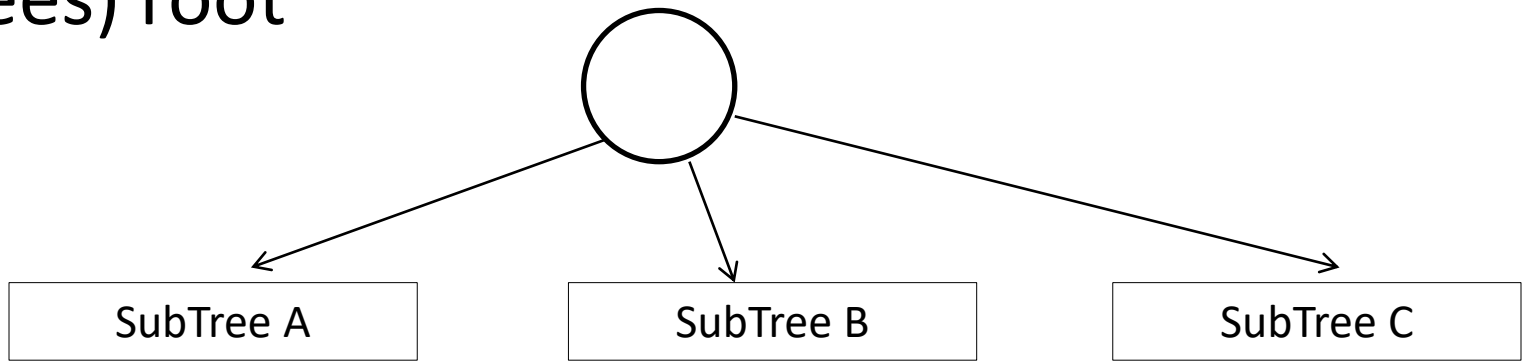
E, F.

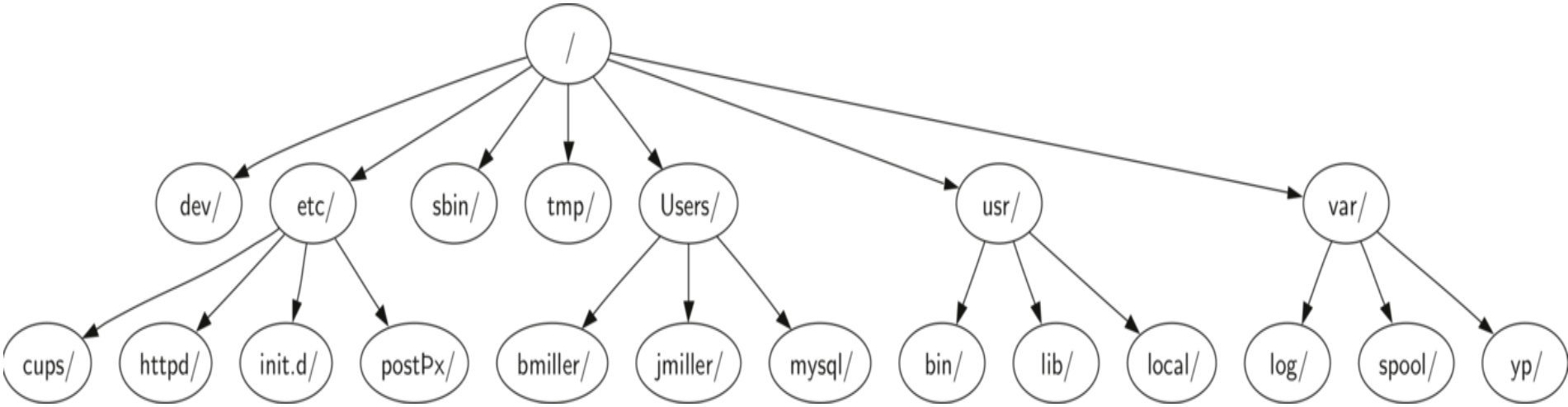
Formal Definition of a Tree

- A tree is either

✓empty (no nodes) or

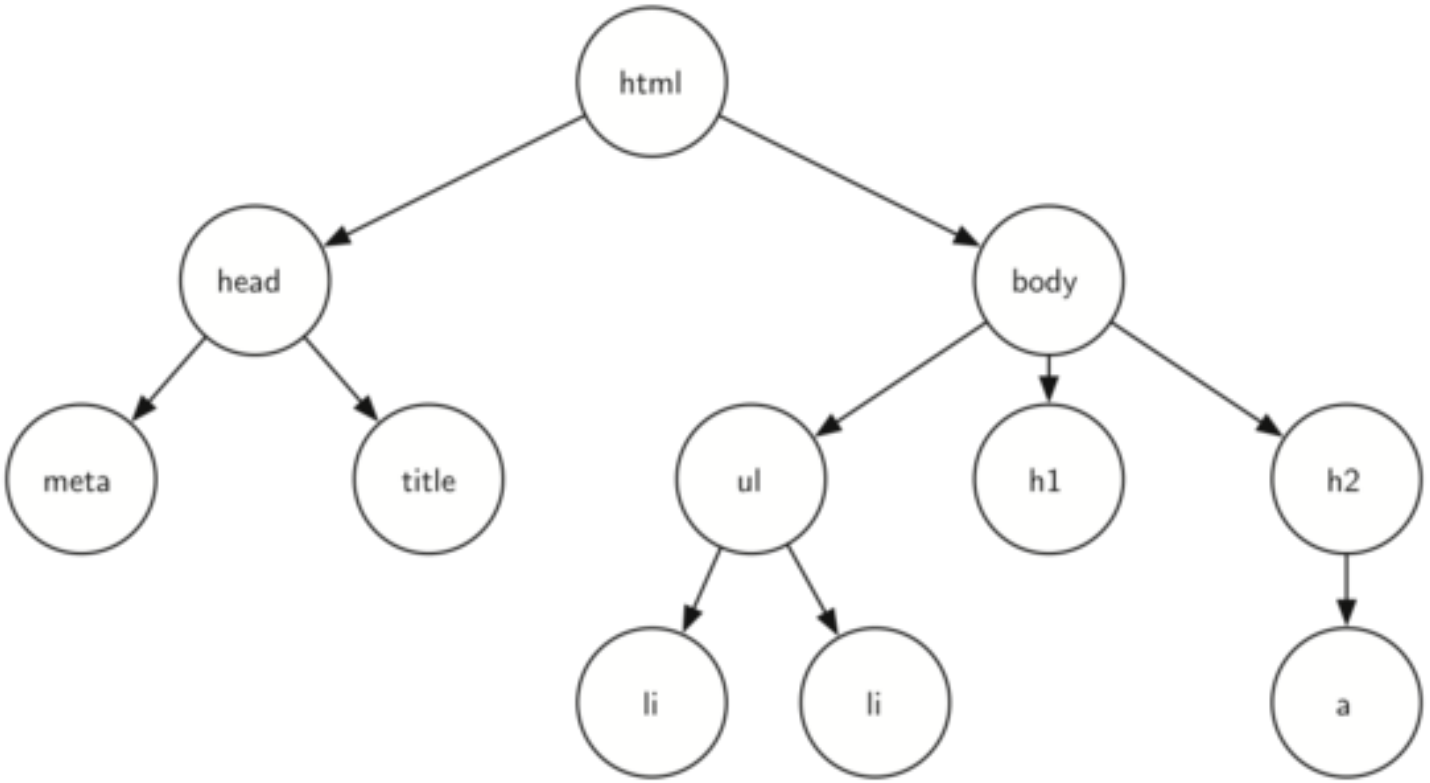
✓a root connected to 0 or more trees (called sub trees) root





A small part of the unix file system hierarchy

```
<html>
<head>
    <title>simple</title>
</head>
<body>
    <h1>A simple web page</h1>
    <ul>
        <li>List item one</li>
        <li>List item two</li>
    </ul>
    <h2><a href="https://www.google.com">Google</a><h2>
</body>
</html>
```



A tree corresponding to the markup elements of a web page

Definitions

Node

A node is a fundamental part of a tree. It can have a unique name, which we sometimes call the "**key.**" A node may also have **additional information**, which we refer to in this book as the "payload." While the payload information is not central to many tree algorithms, it is often critical in applications that make use of trees.

Edge

An edge is another fundamental part of a tree. An edge **connects two nodes** to show that there is a relationship between them. Every node other than the root is connected by exactly one incoming edge from another node. Each node may have several outgoing edges.

Root

The **root of the tree is the only node in the tree that has no incoming edges.** In a file system, / is the root of the tree. In an HTML document, the <html> tag is the root of the tree.

Path

A path is an **ordered list of nodes that are connected by edges**.

Children

The set of nodes that have **incoming edges from the same node are said to be the children of that node**. In our file system example, nodes log/, spool/, and yp/ are the children of node var/.

Parent

A node is the parent of all the nodes to which it connects with outgoing edges. In our file system example the node var/ is the parent of nodes log/, spool/, and yp/.

Sibling

Nodes in the tree that are children of the same parent are said to be siblings. The nodes etc/ and usr/ are siblings in the file system tree.

Subtree

A subtree is a set of nodes and edges comprised of a parent and all the descendants of that parent.

Leaf Node

A leaf node is a node that has no children. For example, Human and Chimpanzee are leaf nodes in our animal taxonomy example.

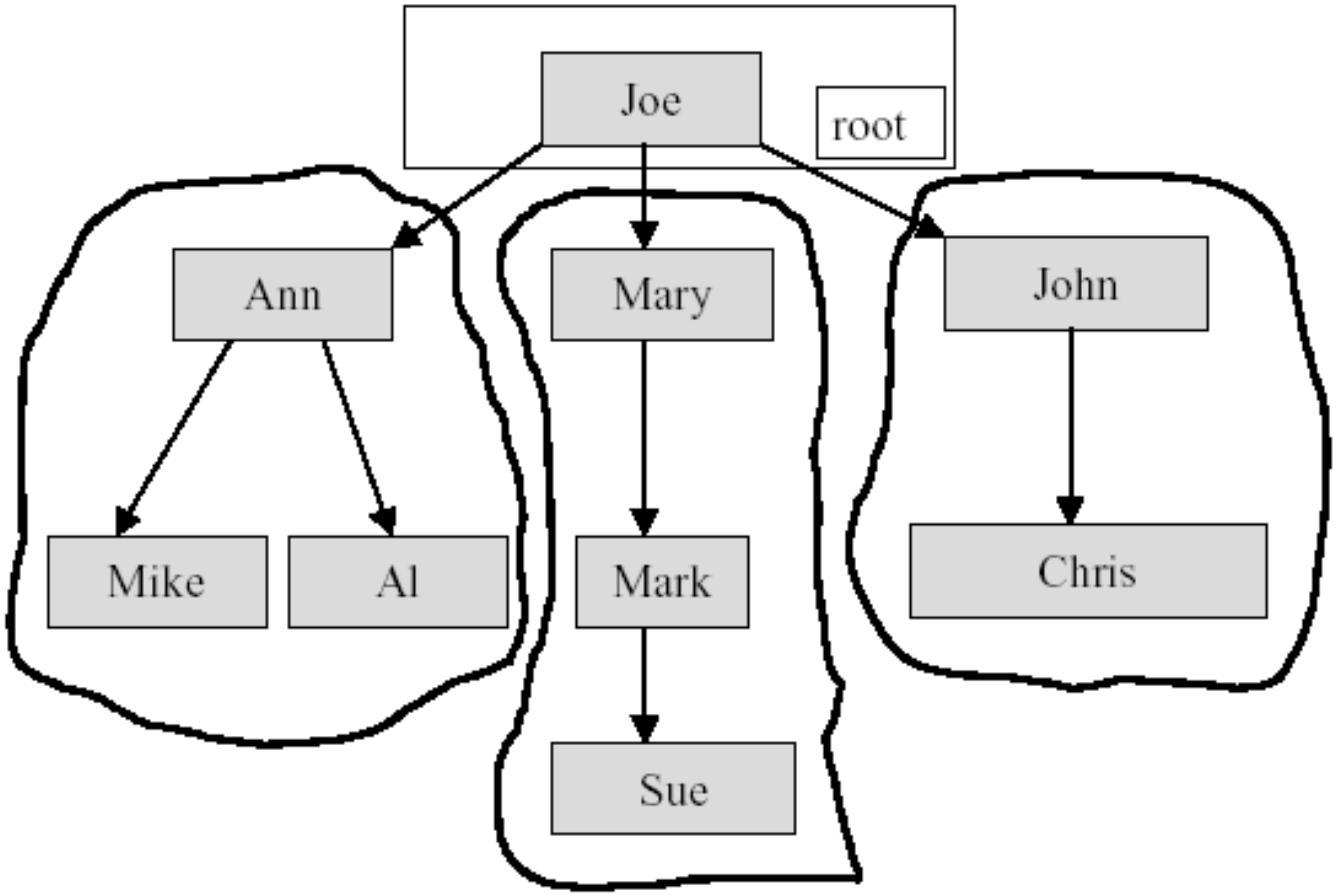
Level

The **level of a node *n* is the number of edges on the path** from the root node to *n*. For example, the level of the Felis node in our animal taxonomy example is five. By definition, the level of the root node is zero.

Height

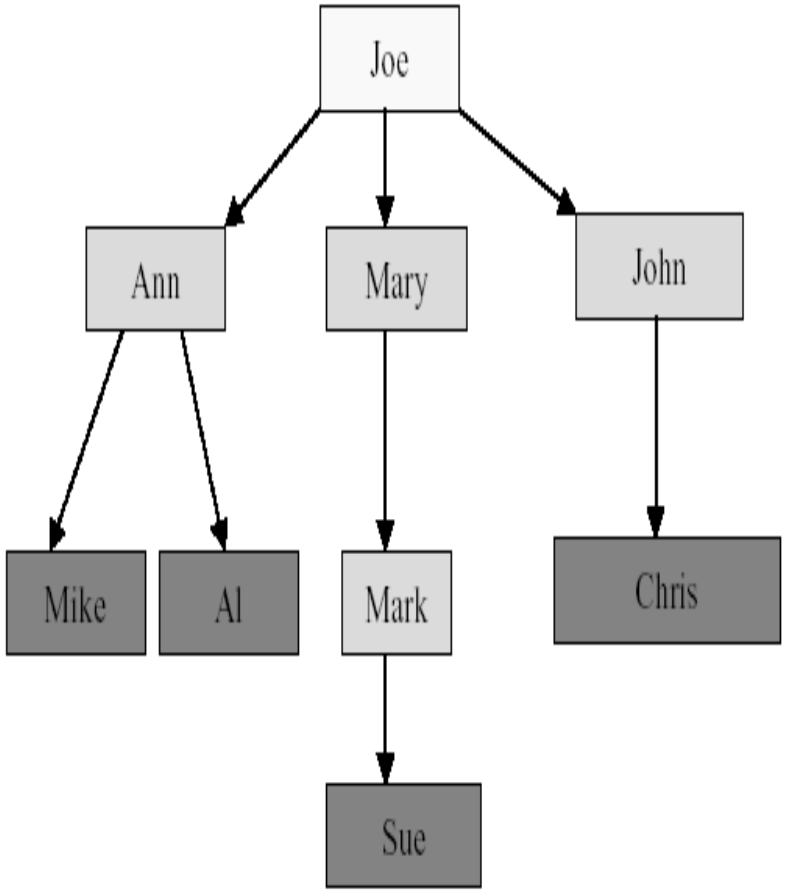
The **height of a tree is equal to the maximum level of any node in the tree.** The height of the tree in our file system example is two.

Subtrees



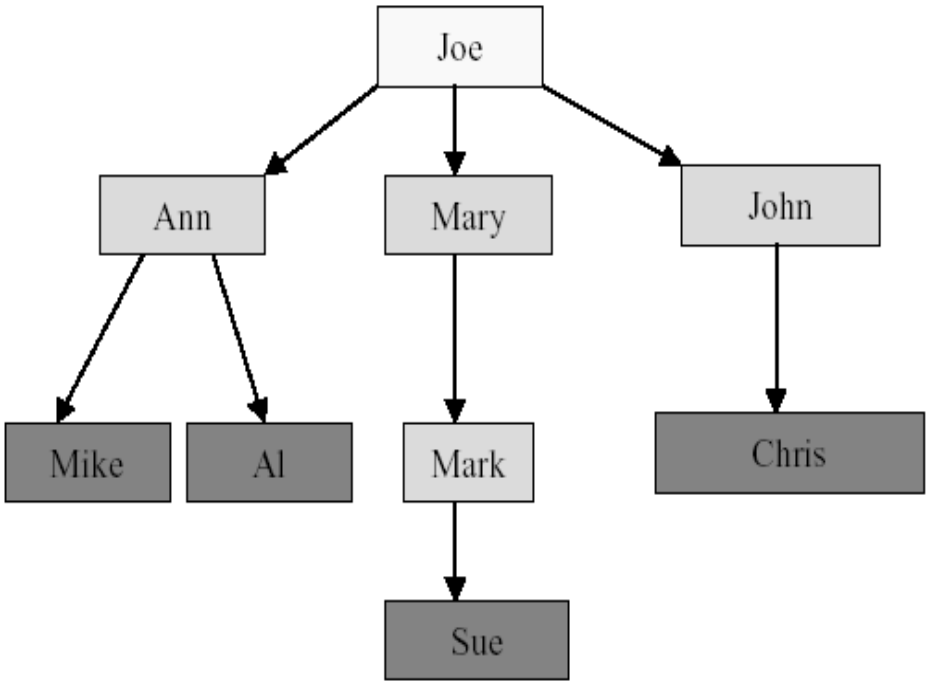
Tree Terminology

- 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**.



Other Definitions

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



Leaves = {Mike,Al,Sue,Chris}

Parent(Mary) = Joe

Grandparent(Sue) = Mary

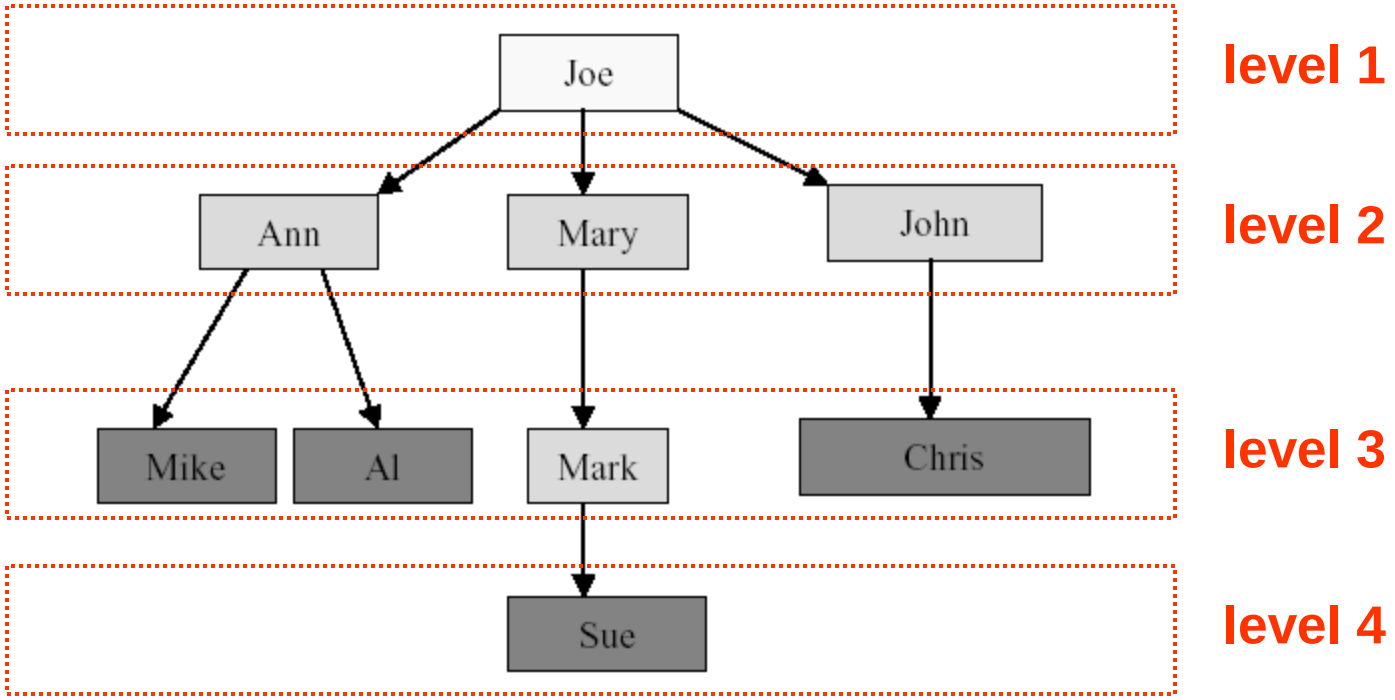
Siblings(Mary) = {Ann,John}

Ancestors(Mike) = {Ann,Joe}

Descendents(Mary)={Mark,Sue}

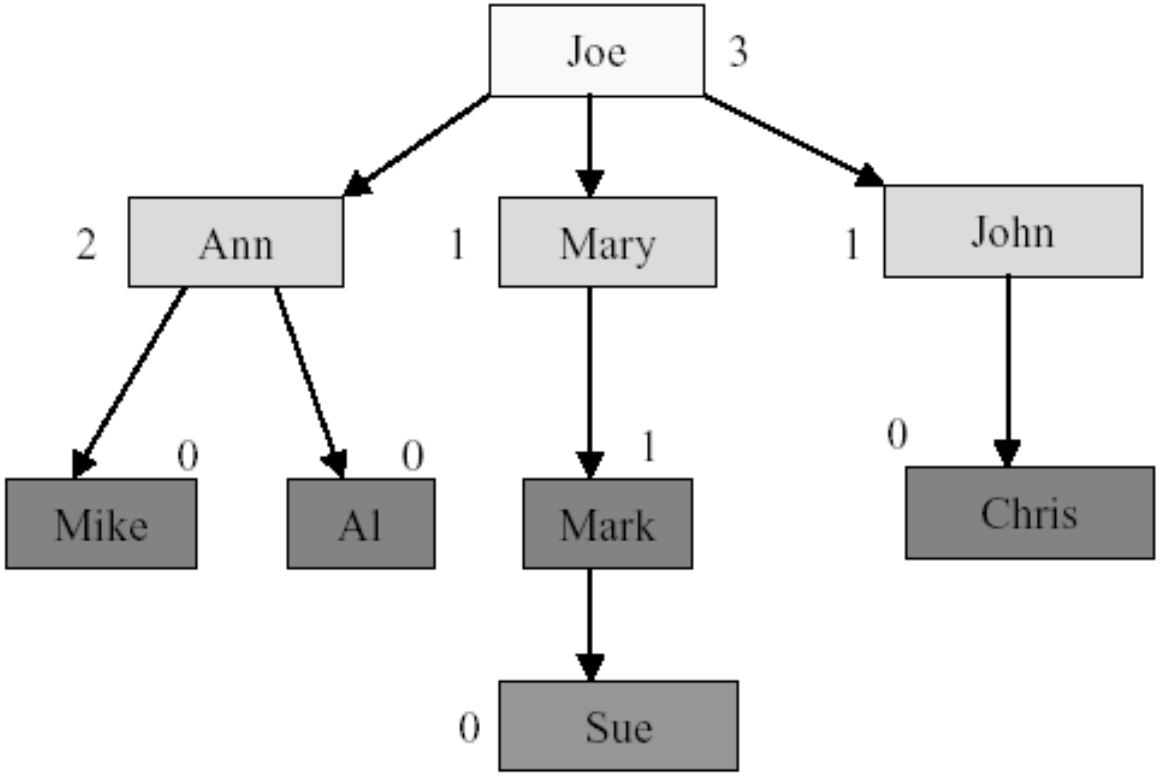
Levels and Height

- Root is at level 1 and its children are at level 2.
- Height = depth = number of levels



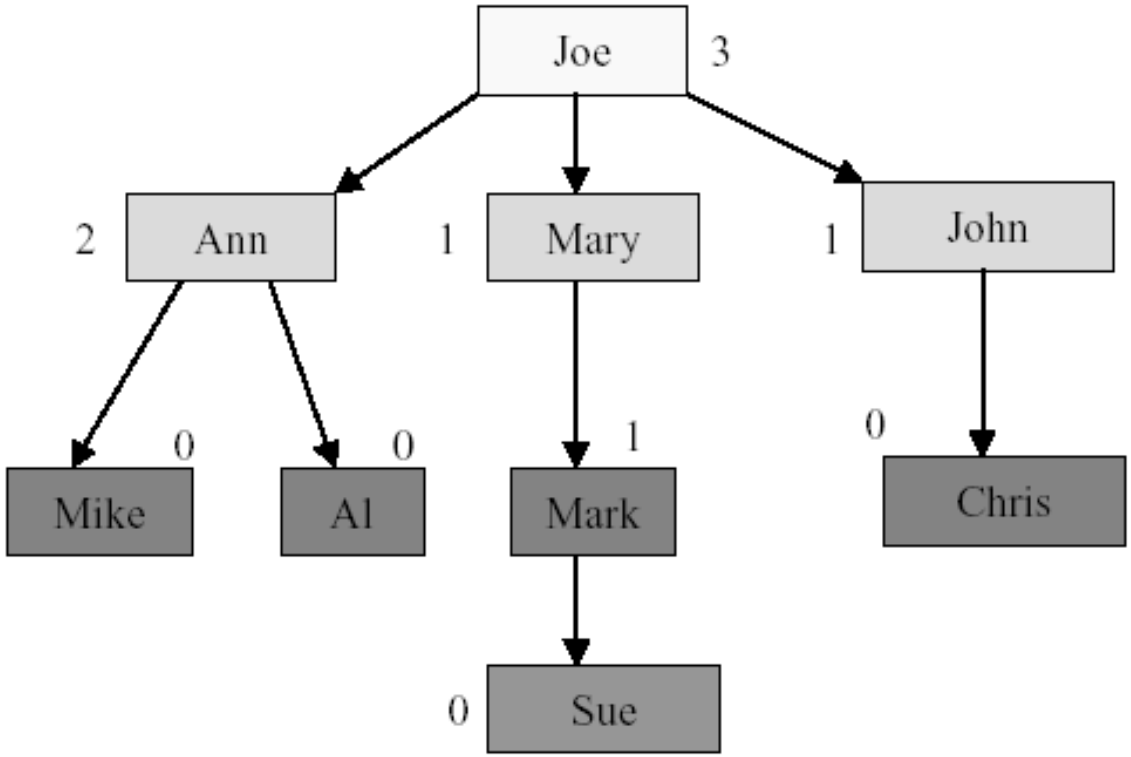
Node Degree

- Node degree is the number of children it has



Tree Degree

- Tree degree is the maximum of node degrees



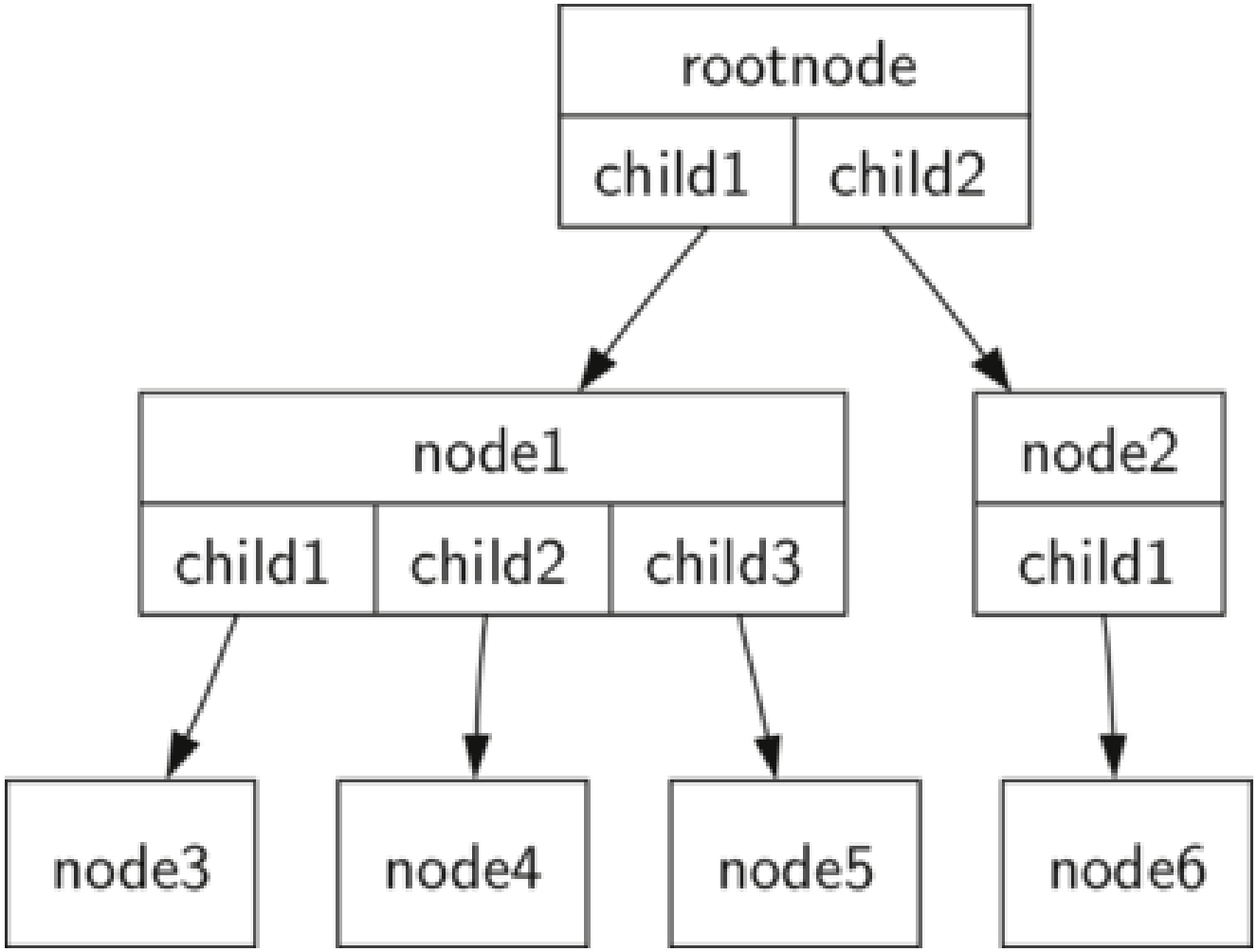
tree degree = 3

With the basic vocabulary now defined, we can move on to two formal definitions of a tree: one involving nodes and edges, and the other a recursive definition.

Definition one: A tree consists of a set of nodes and a set of edges that connect pairs of nodes. A tree has the following properties:

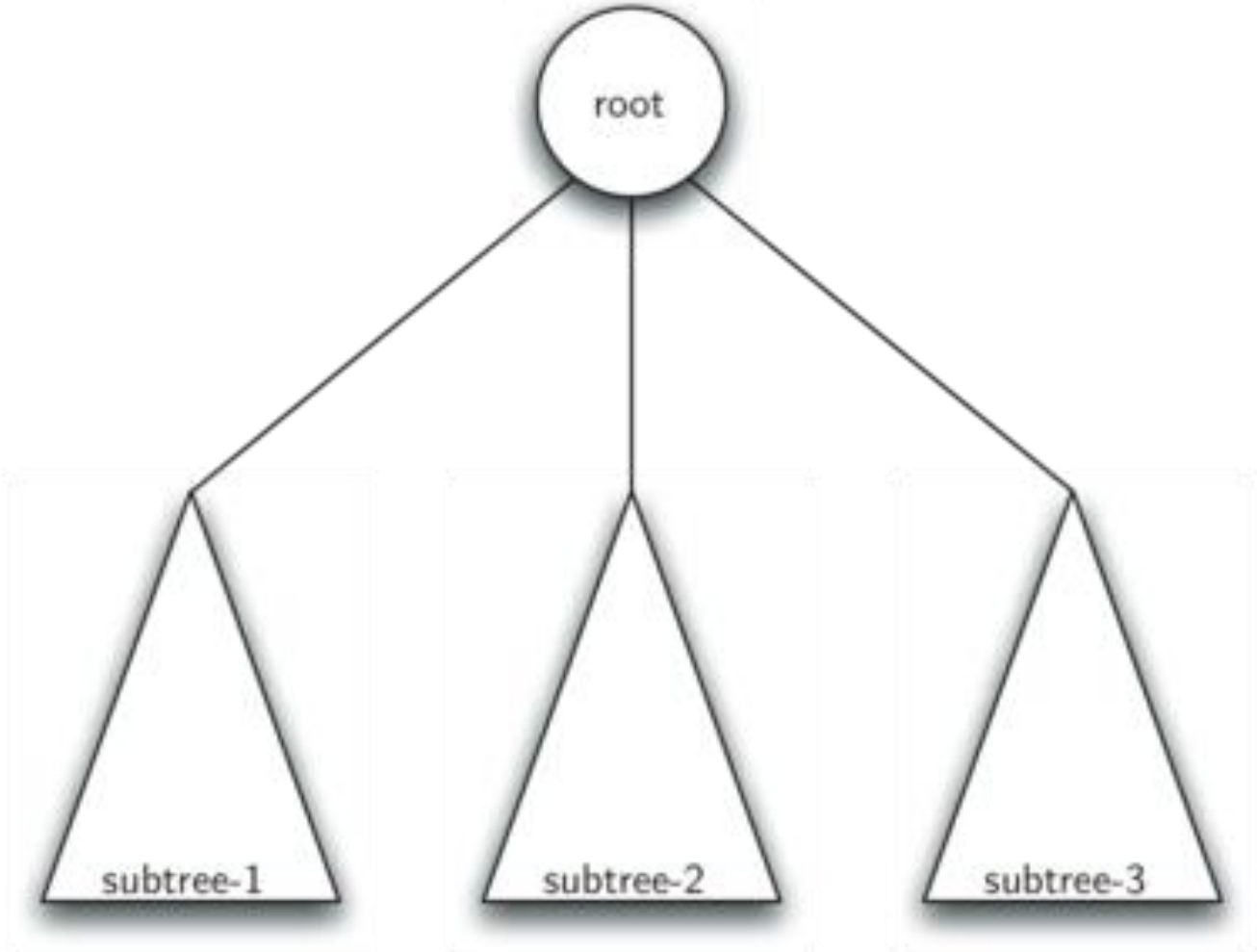
- One node of the tree is designated as the root node.
- Every node *n*, except the root node, is connected by an edge from exactly one other node *p*, where *p* is the parent of *n*.
- A unique path traverses from the root to each node.
- If each node in the tree has a **maximum of two children**, we say that the tree is a **binary tree**.

The diagram below illustrates a tree that fits definition one. The arrowheads on the edges indicate the direction of the connection.



Definition two: **A tree is either empty or consists of a root and zero or more subtrees, each of which is also a tree. The root of each subtree is connected to the root of the parent tree by an edge.**

The diagram below illustrates this recursive definition of a tree. Using the recursive definition of a tree, we know that the tree below has at least four nodes, since each of the triangles representing a subtree must have a root. It may have many more nodes than that, but we do not know unless we look deeper into the tree.


root
subtree-1
subtree-2
subtree-3

Levels of a node

Levels of a node represents the number of connections between the node and the root. It represents generation of a node. If the root node is at level 0, its next node is at level 1, its grand child is at level 2 and so on. Levels of a node can be shown as follows:

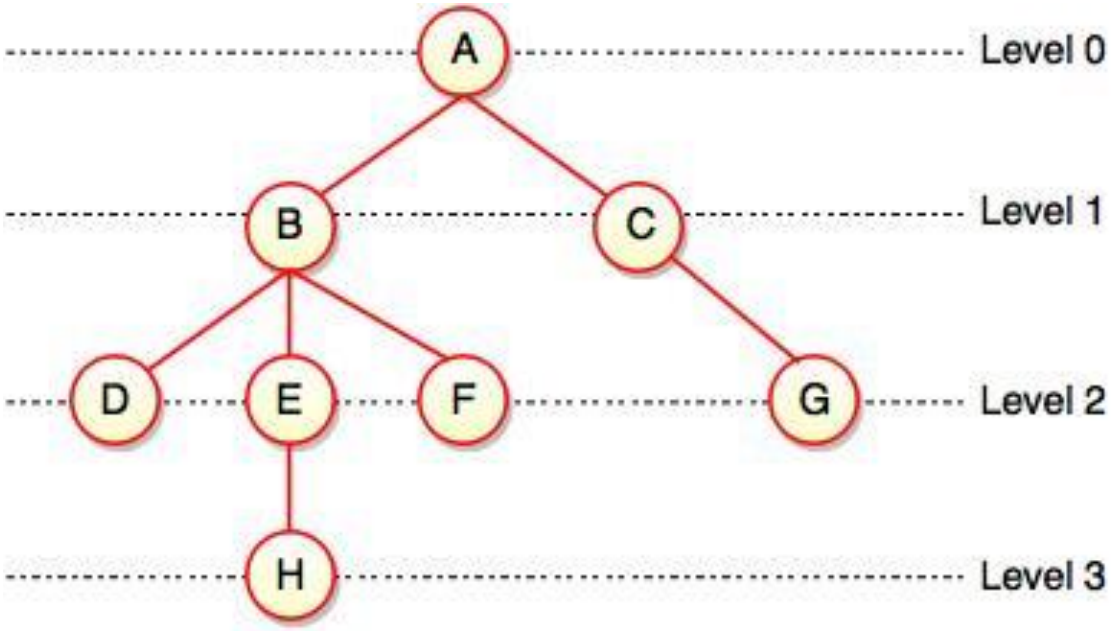


Fig. Levels of Tree

Note:

- If node has no children, it is called **Leaves** or **External Nodes.**
- Nodes which are not leaves, are called **Internal Nodes**. Internal nodes have at least one child.
- A tree can be empty with no nodes or a tree consists of one node called the **Root**.

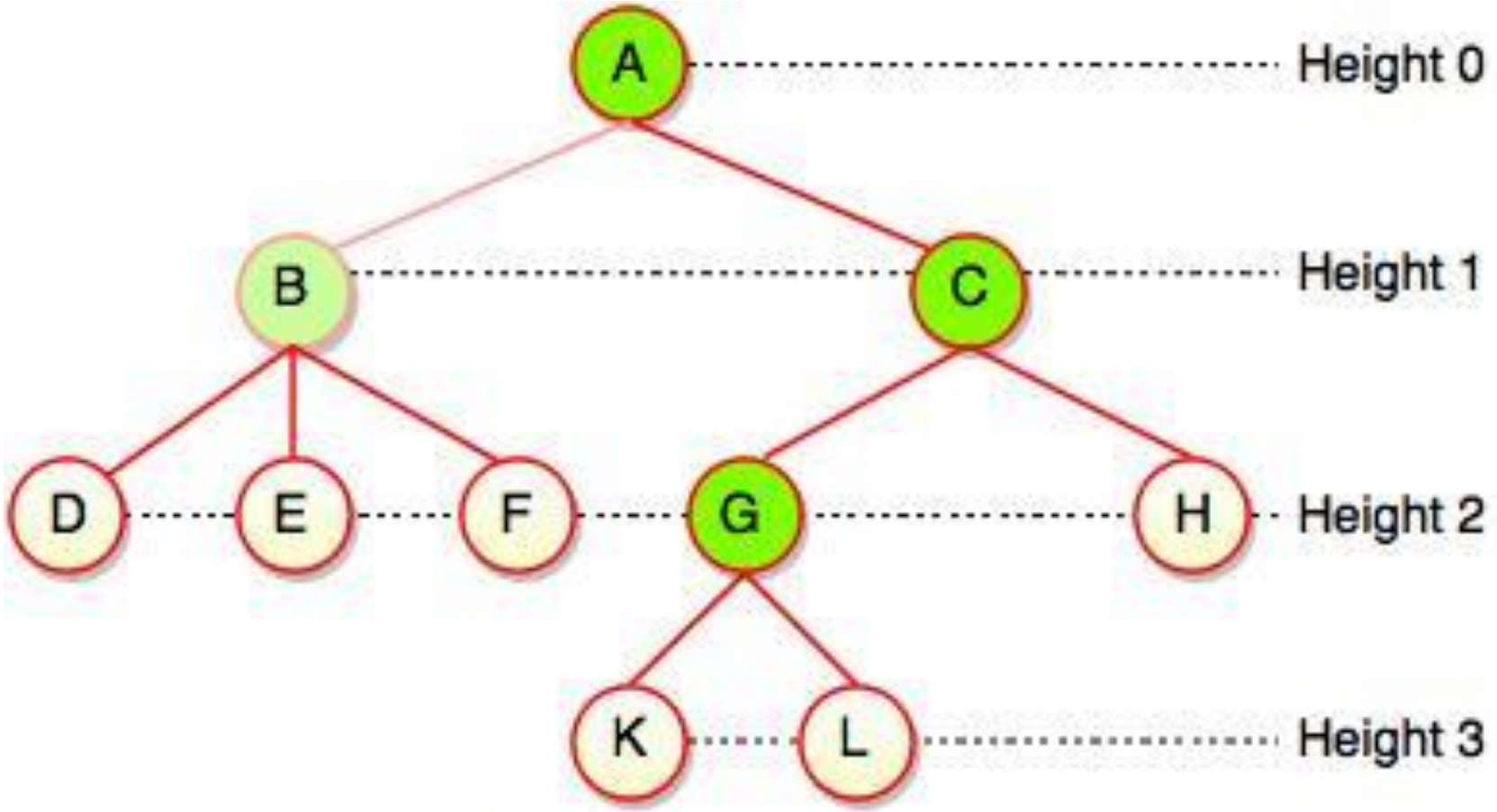


Fig. Height of a Node

As we studied, height of a node is a number of edges on the longest path between that node and a leaf. Each node has height.

In the above figure, A, B, C, D can have height. Leaf cannot have height as there will be no path starting from a leaf. Node A's height is the number of edges of the path to K not to D. And its height is 3.

Note:

- Height of a node defines the longest path from the node to a leaf.
- Path can only be downward.

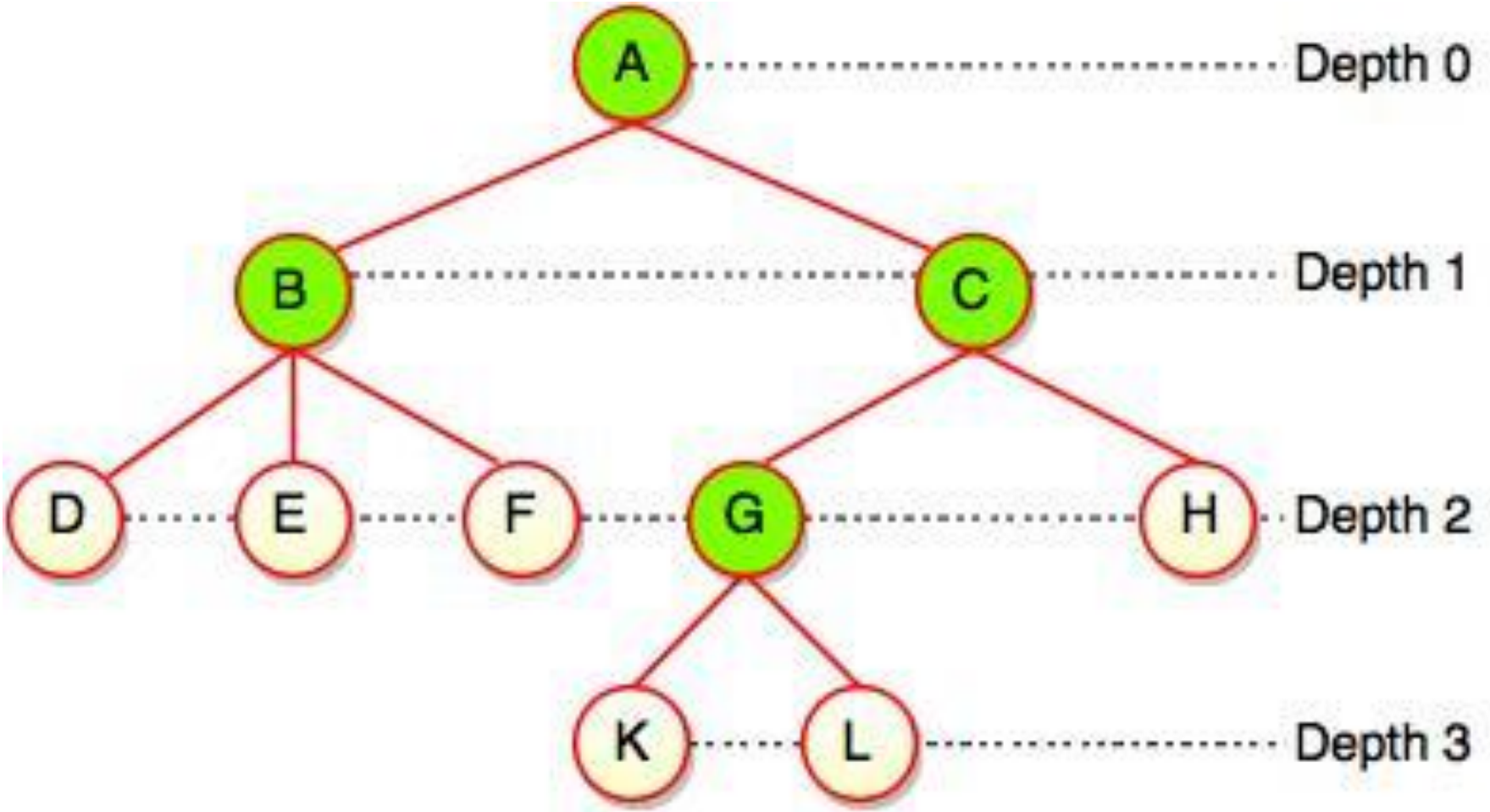


Fig. Depth of a Node

While talking about the height, it locates a node at bottom where for depth, it is located at top which is root level and therefore we call it depth of a node.

In the above figure, Node G's depth is 2. In depth of a node, we just count how many edges between the targeting node & the root and ignoring the directions.

Note: Depth of the root is 0.

Advantages of Tree

- Tree **reflects structural relationships** in the data.
- It is used to **represent hierarchies**.
- It provides an **efficient insertion and searching operations.**
- Trees are flexible. It allows to move **subtrees around with minimum effort.**

Tree is a collection of elements called Nodes, where each node can have arbitrary number of children.

Field	Description
Root	Root is a special node in a tree. The entire tree is referenced through it. It does not have a parent.
Parent Node	Parent node is an immediate predecessor of a node.
Child Node	All immediate successors of a node are its children.
Siblings	Nodes with the same parent are called Siblings.
Path	Path is a number of successive edges from source node to destination node.
Height of Node	Height of a node represents the number of edges on the longest path between that node and a leaf.
Height of Tree	Height of tree represents the height of its root node.
Depth of Node	Depth of a node represents the number of edges from the tree's root node to the node.
Degree of Node	Degree of a node represents a number of children of a node.
Edge	Edge is a connection between one node to another. It is a line between two nodes or a node and a leaf.

Binary Tree

Binary Tree

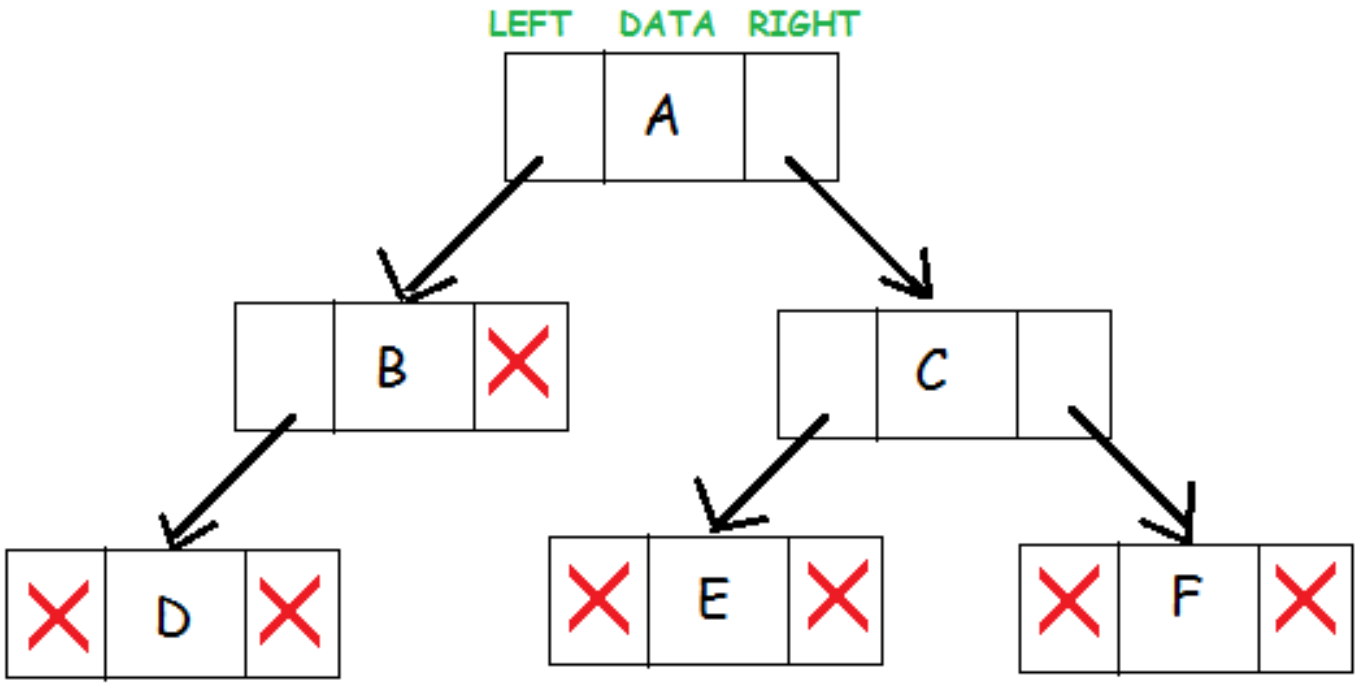
- A finite (possibly empty) collection of elements
- A nonempty binary tree has a root element and the remaining elements (if any) are partitioned into two binary trees
- They are called the left and right subtrees of the binary tree

A binary tree is a hierarchical data structure in which each **node has at most two children generally referred as left child and right child.**

- Each node contains three components:
  - Reference to left subtree
  - Reference to right subtree
  - Data element

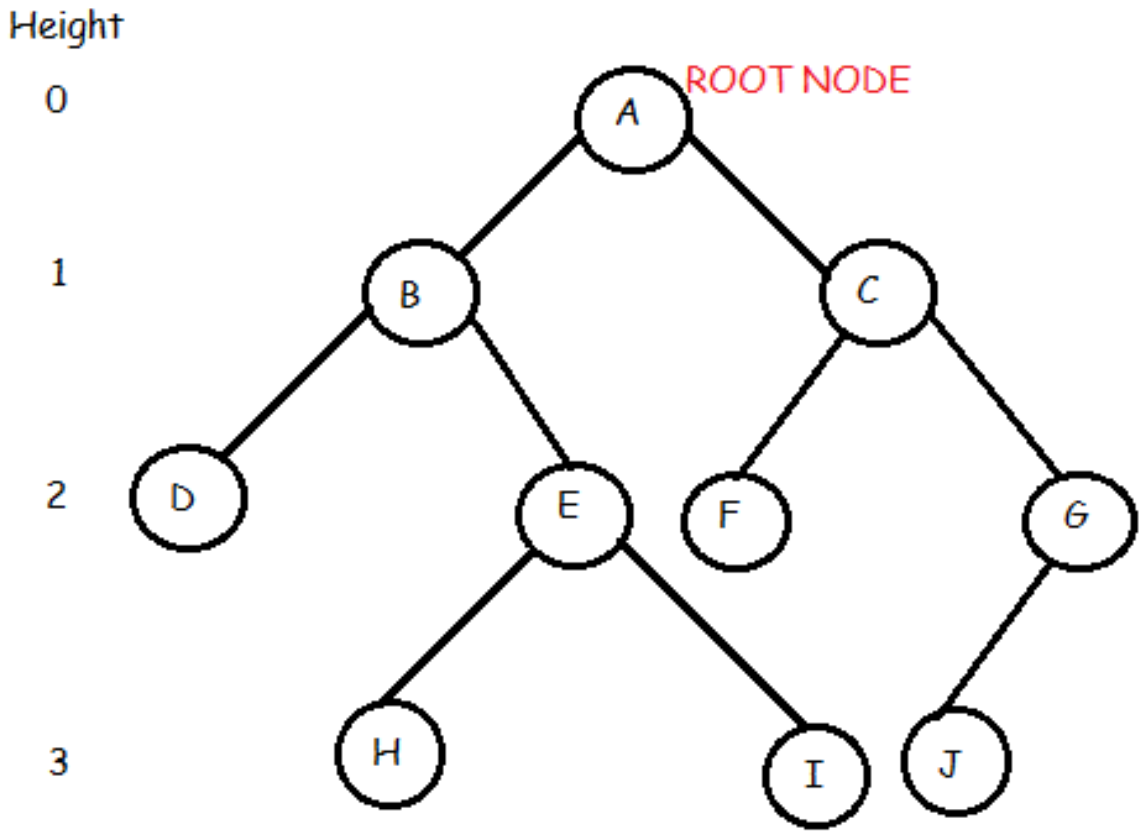
The topmost node in the tree is called the root. An empty tree is represented by **NULL** pointer.

A representation of binary tree is shown:



Binary Tree: Common Terminologies

- **Root:** Topmost node in a tree.
- **Parent:** Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent.
- **Child:** A node directly connected to another node when moving away from the root.
- **Leaf/External node:** Node with no children.
- **Internal node:** Node with atleast one children.
- **Depth of a node:** Number of edges from root to the node.
- **Height of a node:** Number of edges from the node to the deepest leaf. Height of the tree is the height of the root.



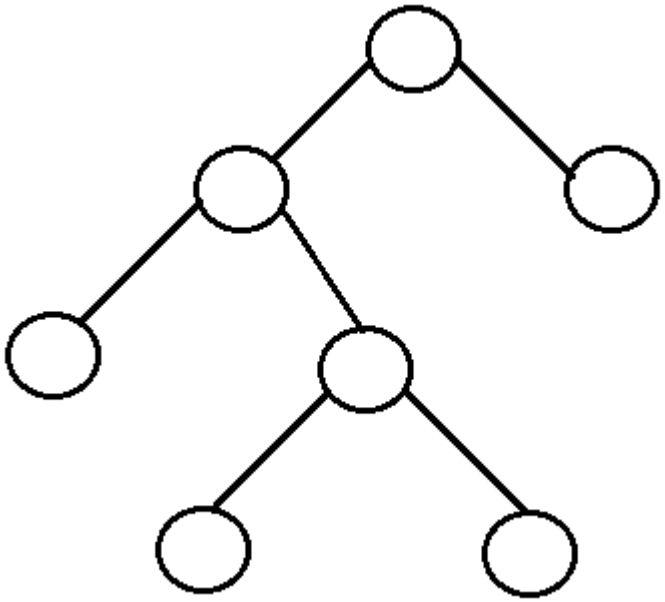
In the above binary tree we see that root node is **A**. The tree has 10 nodes with 5 internal nodes, i.e, **A,B,C,E,G** and 5 external nodes, i.e, **D,F,H,I,J**. The height of the tree is 3. **B** is the parent of **D** and **E** while **D** and **E** are children of **B**.

Advantages of Trees

- Trees are so useful and frequently used, because they have some very serious advantages:
- Trees reflect structural relationships in the data.
- Trees are used to represent hierarchies.
- Trees provide an efficient insertion and searching.
- Trees are very flexible data, allowing to move subtrees around with minimum effort.

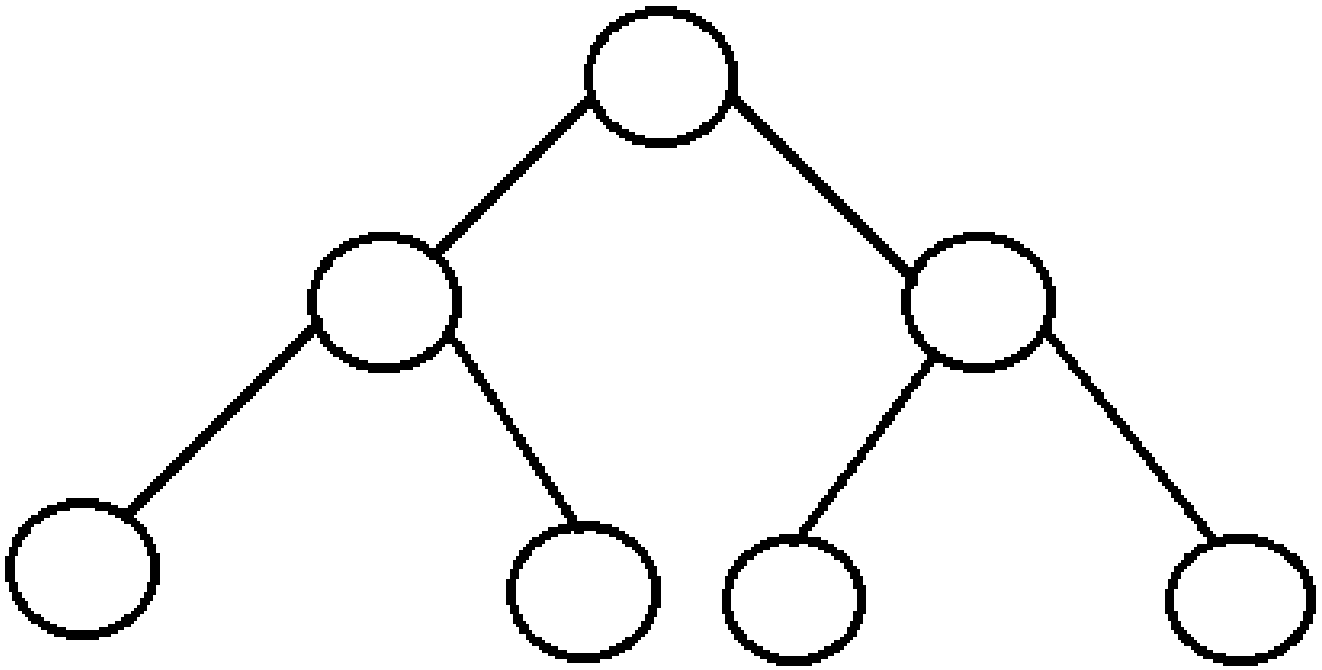
Types of Binary Trees (Based on Structure)

- **Rooted binary tree:** It **has a root node and every node has at most two children.**
- **Full binary tree:** It is a tree in **which every node in the tree has either 0 or 2 children.**

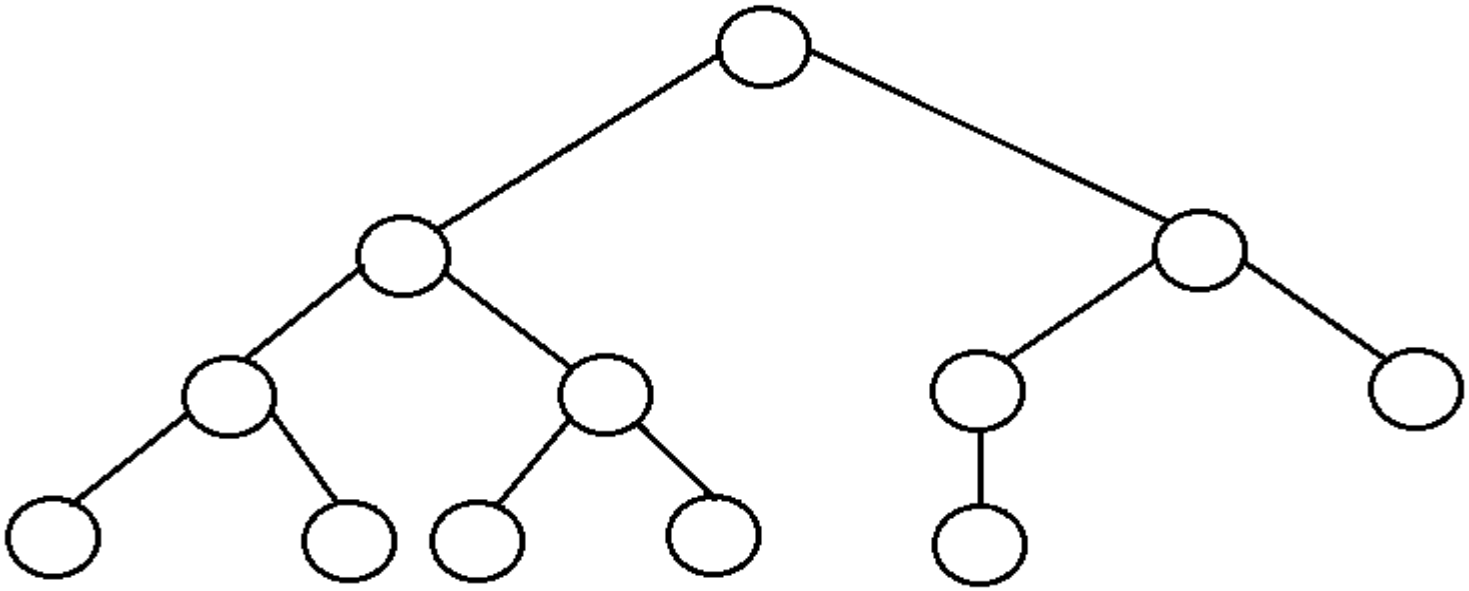


- The number of nodes, *n*, in a full binary tree is at least n = 2h – 1, and atmost $n = 2^{h+1} - 1$, where *h* is the height of the tree.
- The number of leaf nodes *l*, in a full binary tree is number, *L* of internal nodes + 1, i.e, $l = L+1$.

Perfect binary tree: It is a binary tree in which all **interior nodes have two children** and **all leaves have the same depth or same level.**



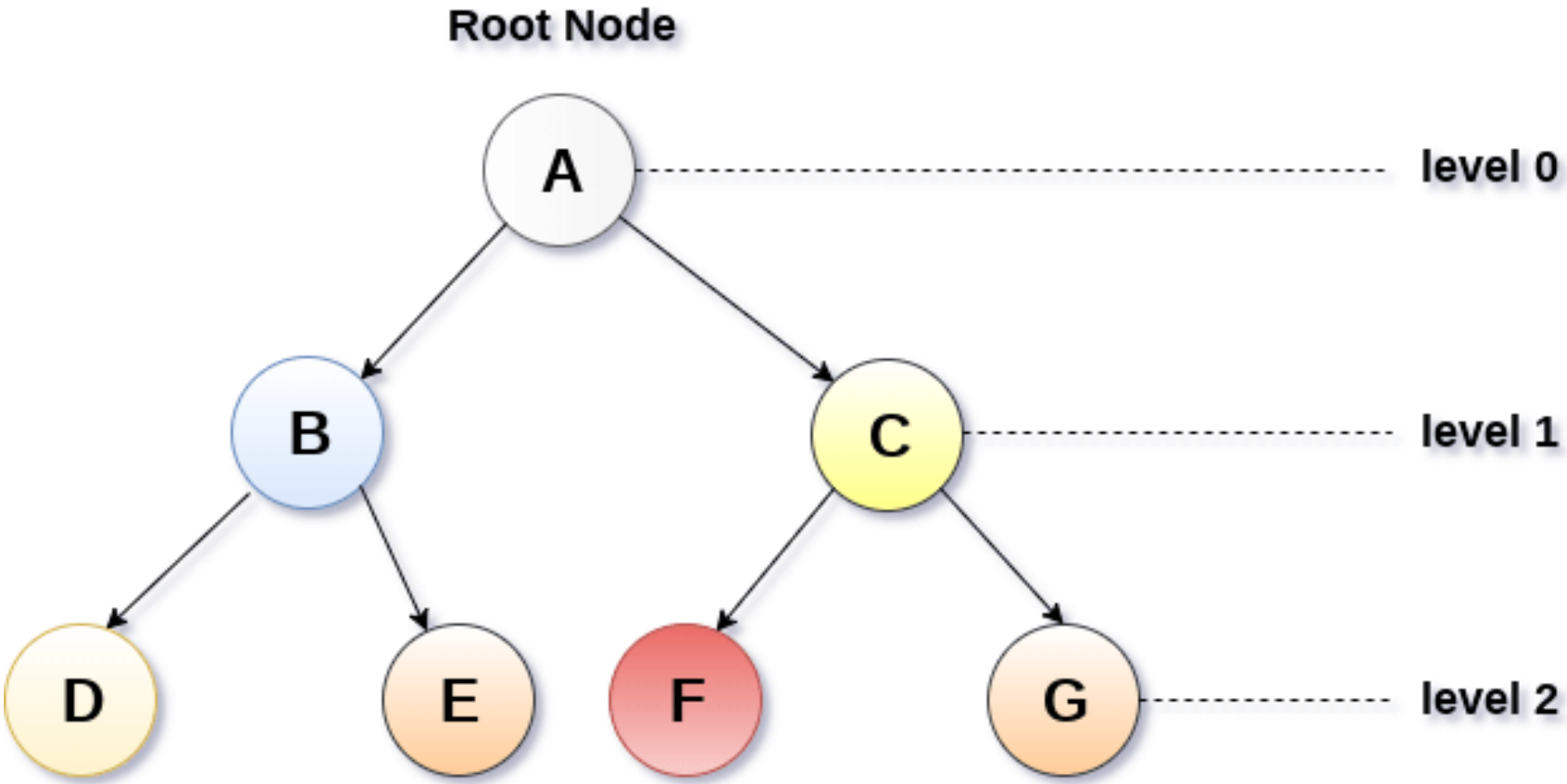
Complete binary tree: It is a binary tree in **which every level, except possibly the last**, **is completely filled**, and **all nodes are as far left as possible.**



The **number of internal nodes** in a complete binary tree of n nodes is ***floor(n/2)***.

Complete Binary Tree

A Binary Tree is said to be a complete binary tree if **all of the leaves are located at the same level d**. A **complete binary tree is a binary tree that contains exactly 2^l nodes at each level between level 0 and d.** The **total number of nodes in a complete binary tree with depth d is 2^{d+1}-1** where **leaf nodes are 2^d while non-leaf nodes are 2^d-1.**

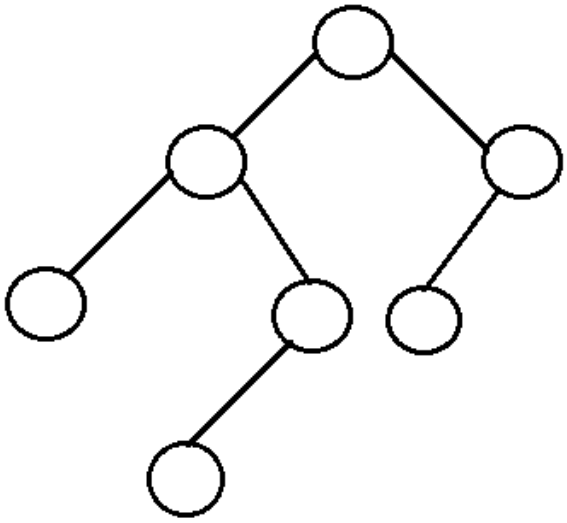


Complete Binary Tree

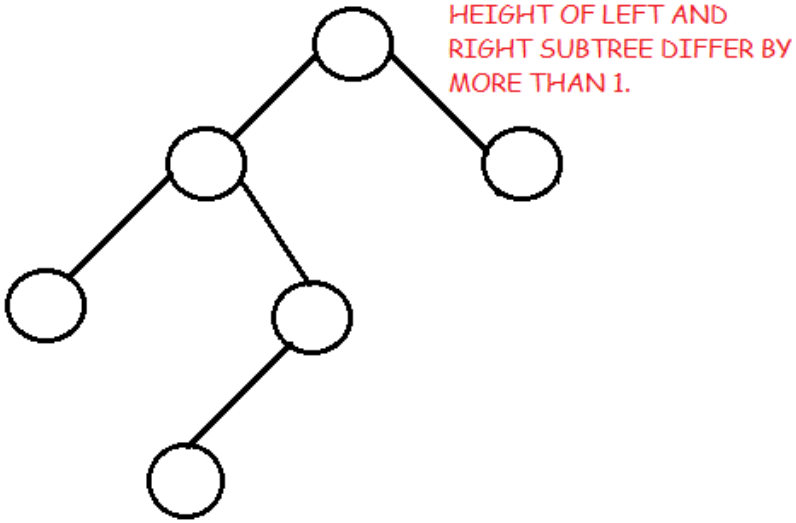
Balanced binary tree: A **binary tree is height balanced** if it satisfies the **following constraints**:

- **The left and right subtrees' heights differ by at most one**, AND
- **The left subtree is balanced**, AND
- The **right subtree is balanced**

An empty tree is height balanced.



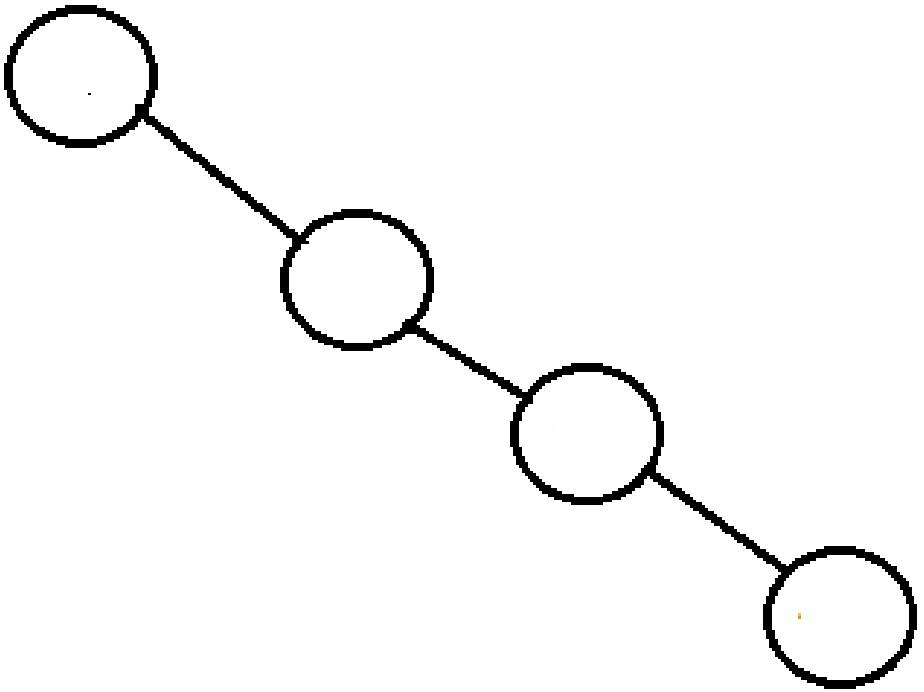
HEIGHT BALANCED BINARY TREE



NOT A HEIGHT BALANCED BINARY TREE

The height of a balanced binary tree is O(*Log n*) where *n* is number of nodes.

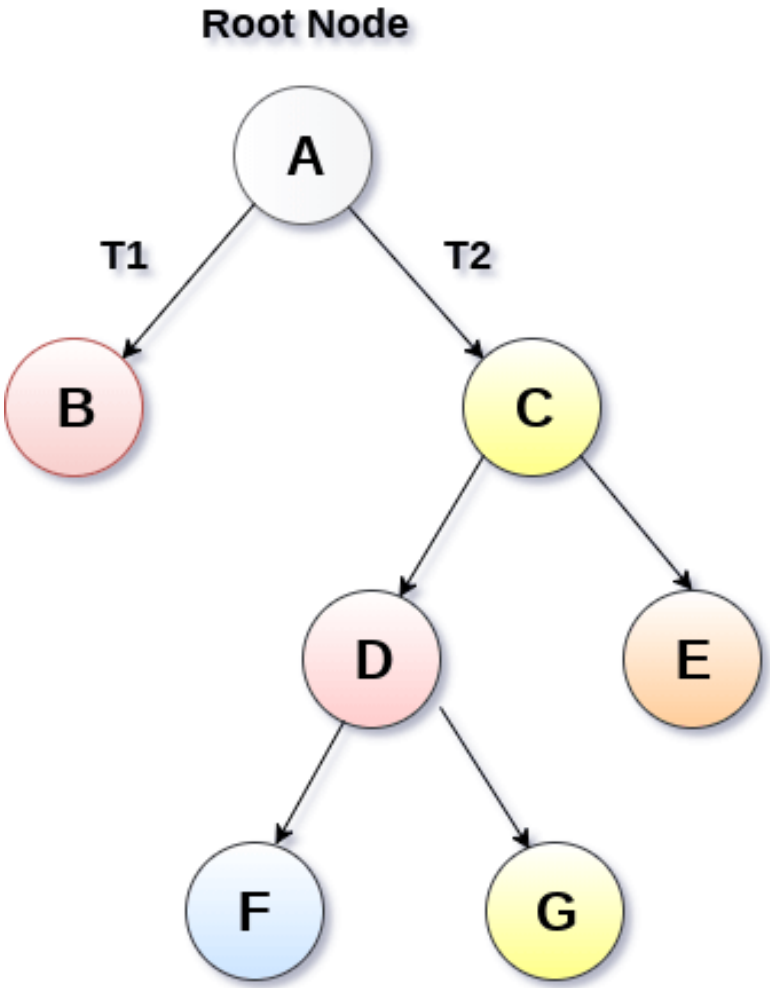
Degenarate tree: It is a tree is **where each parent node has only one child node**. It behaves like a linked list.



Strictly Binary Tree

In Strictly Binary Tree, **every non-leaf node contain non-empty left and right sub-trees**. In other words, **the degree of every non-leaf node will always be 2**. A strictly binary tree with n leaves, will have (2n - 1) nodes.

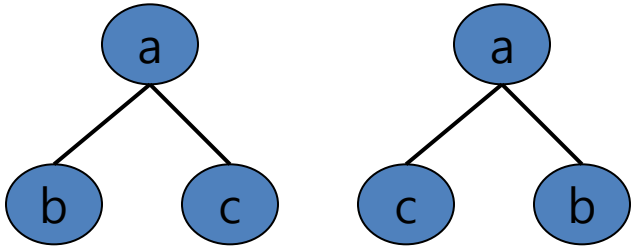
A strictly binary tree is shown in the following figure.



Strictly Binary Tree

Difference Between a Tree & a Binary Tree

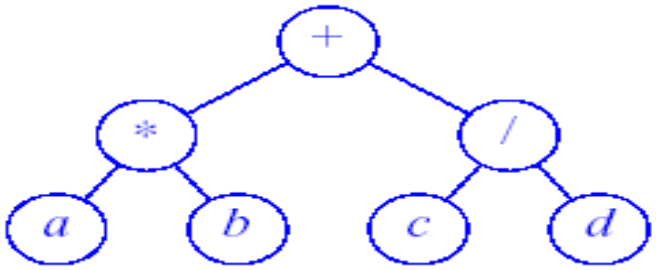
- A binary tree may be empty; a tree cannot be empty.
- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree.
- The subtrees of a binary tree are ordered; those of a tree are not ordered.



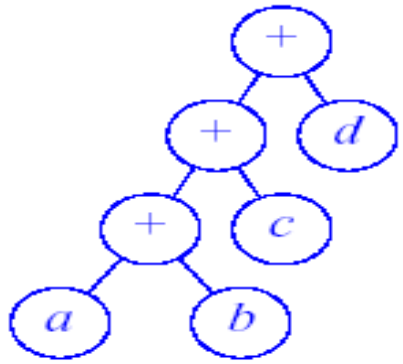
- different when viewed as a binary tree

- same when viewed as a tree

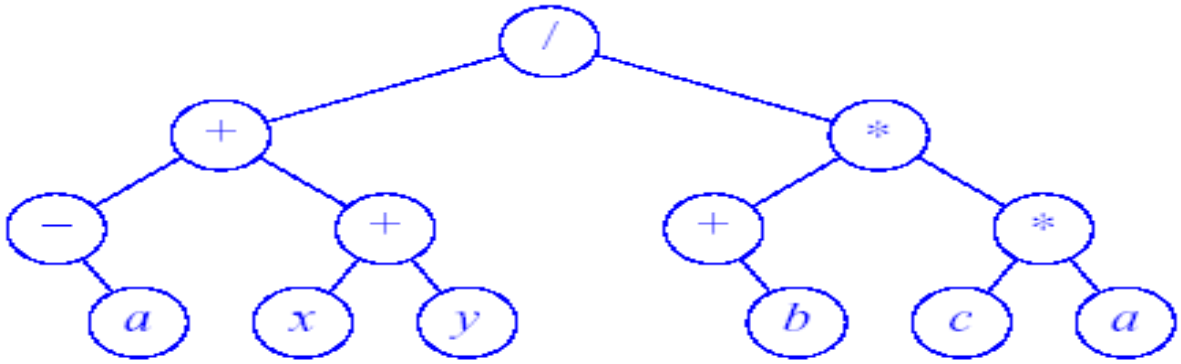
Binary Tree for Expressions



(a) $(a * b) + (c / d)$



(b) $((a + b) + c) + d)$



(c) $((-a) + (x + y)) / ((+b) * (c * a))$

Expression Trees

Binary Tree Properties

The drawing of every binary tree with n elements, n > 0, has exactly n-1 edges.

let us consider the binary tree with n nodes, to find the number of edges, we start from the root node- taking complete binary tree as example

- root node has two edges attached to its children
- internal node has two edges which is attached to its children
- leaf node has no edge attached to its children

now if we add all the edge count taken above:-

2*1(root node) + 2*(n-1/2 -1)(internal node excluding root node) +

0*(n+1/2)(leaf nods)

= n-1(no of edges)

now to generalize:- the above result:-

let us have just binary tree not necessarily complete binary tree:-

we can always construct the complete binary tree with the given binary tree by adding complete sub binary tree,

Important:- all the sub tree added to make a binary tree to complete binary tree is itself complete binary tree. and the root of the sub complete binary tree belongs to given binary tree

so to get the edge count we can get the edge count of constructed complete binary tree and to this subtracting the edge count of the added sub complete binary tree. still you will get no of edges as n-1, where n is number of nodes is binary tree.

A binary tree of height h, h >= 0, has at least h and at most 2^h-1 elements in it.

Here height of a tree is maximum number of nodes on root to leaf path. Height of a tree with single node is considered as 1. This result can be derived from point 2 above. A tree has maximum nodes if all levels have maximum nodes. So maximum number of nodes in a binary tree of height h is $1 + 2 + 4 + .. + 2^{h-1}$. This is a simple geometric series with h terms and sum of this series is $2^h – 1$. In some books, height of the root is considered as 0. In this convention, the above formula becomes $2^{h+1} – 1$

The maximum number of nodes at level 'l' of a binary tree is 2^l.

Here level is number of nodes on path from root to the node (including root and node). Level of root is 0.

This can be proved by induction.

For root, l = 0, number of nodes = $2^0 = 1$

Assume that maximum number of nodes on level 'l' is 2^l

Since in Binary tree every node has at most 2 children, next level would have twice nodes, i.e. $2 * 2^l$

In a Binary Tree with N nodes, minimum possible height or minimum number of levels is $\lfloor Log_2(N+1) \rfloor$

This can be directly derived from point 2 above. If we consider the convention where height of a leaf node is considered as 0, then above formula for minimum possible height becomes

$\lfloor Log_2(N+1) \rfloor - 1$

A Binary Tree with L leaves has at least

$\lfloor Log_2 L \rfloor + 1$ levels

A Binary tree has maximum number of leaves (and minimum number of levels) when all levels are fully filled.

Let all leaves be at level l, then below is true for number of leaves L.

$L <= 2^{l-1}$ [From Point 1]

$l = \lfloor Log_2 L \rfloor + 1$

where l is the minimum number of levels.

In Binary tree where every node has 0 or 2 children, number of leaf nodes is always one more than nodes with two children.

$L = T + 1$

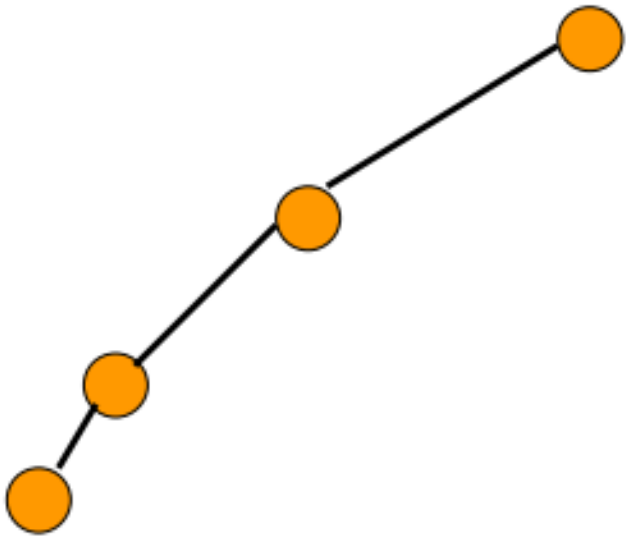
Where L = Number of leaf nodes

T = Number of internal nodes with two children

The height of a binary tree that contains n elements, n >= 0, is <u>at least</u> $\lfloor(\log_2(n+1))\rfloor$ and <u>at most</u> n.

Minimum Number Of Nodes

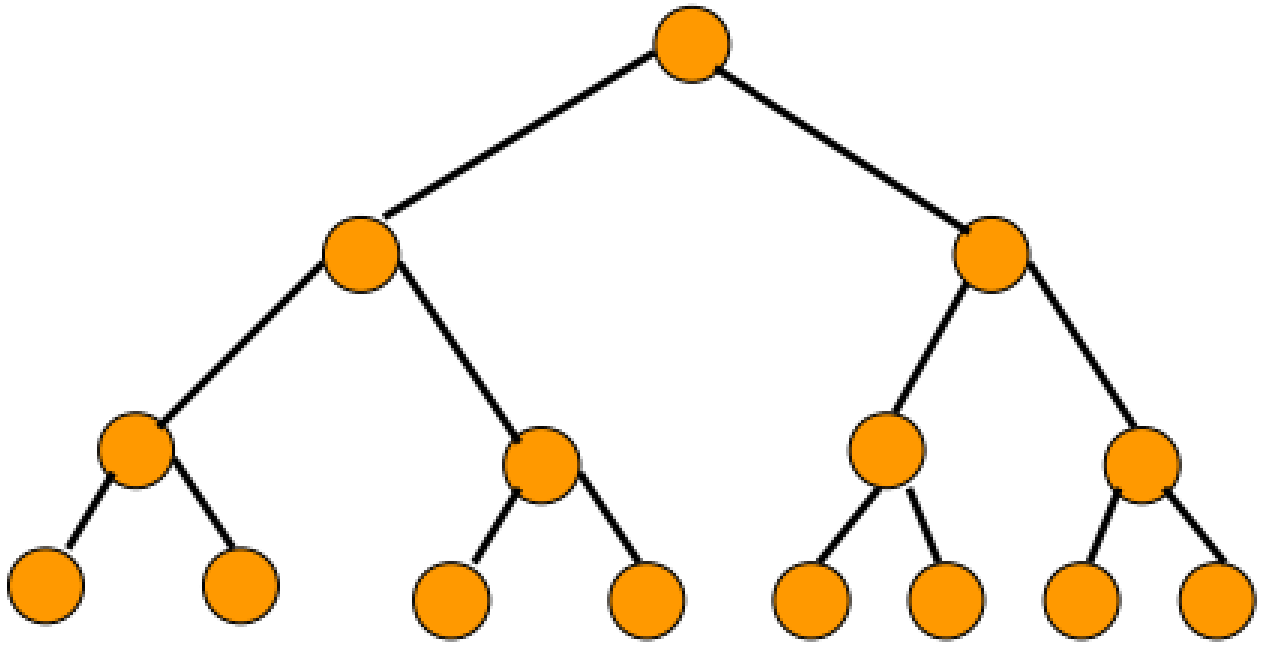
- Minimum number of nodes in a binary tree whose height is h.
- At least one node at each of first h levels.



minimum number of nodes is h

Maximum Number Of Nodes

- All possible nodes at first h levels are present.



Maximum number of nodes

$= 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$

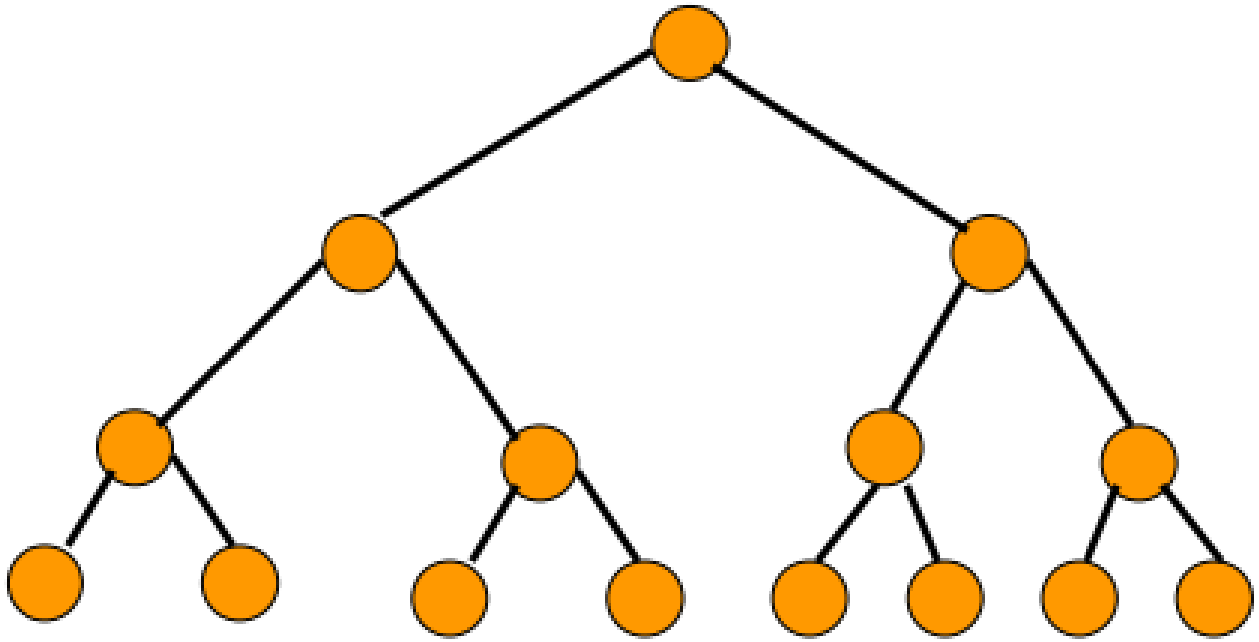
$= 2^{h} - 1$

Number Of Nodes & Height

- Let n be the number of nodes in a binary tree whose height is h.
- $h <= n <= 2^h - 1$
- $\log_2(n+1) <= h <= n$

Full Binary Tree

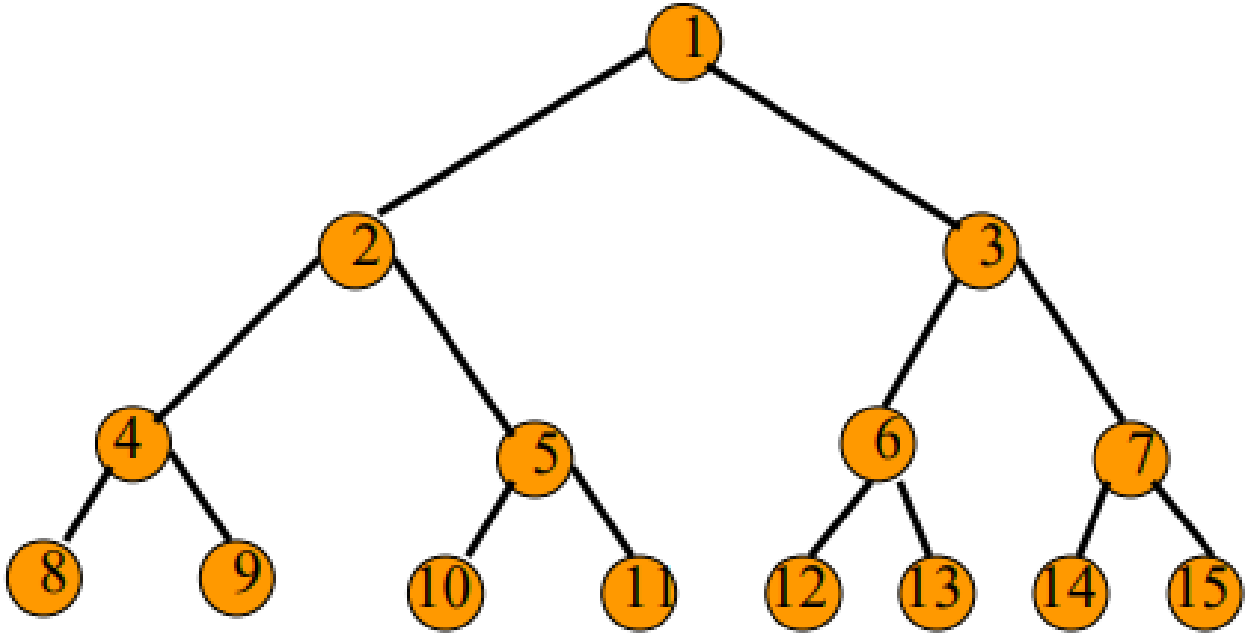
- A full binary tree of a given height h has $2^h - 1$ nodes.



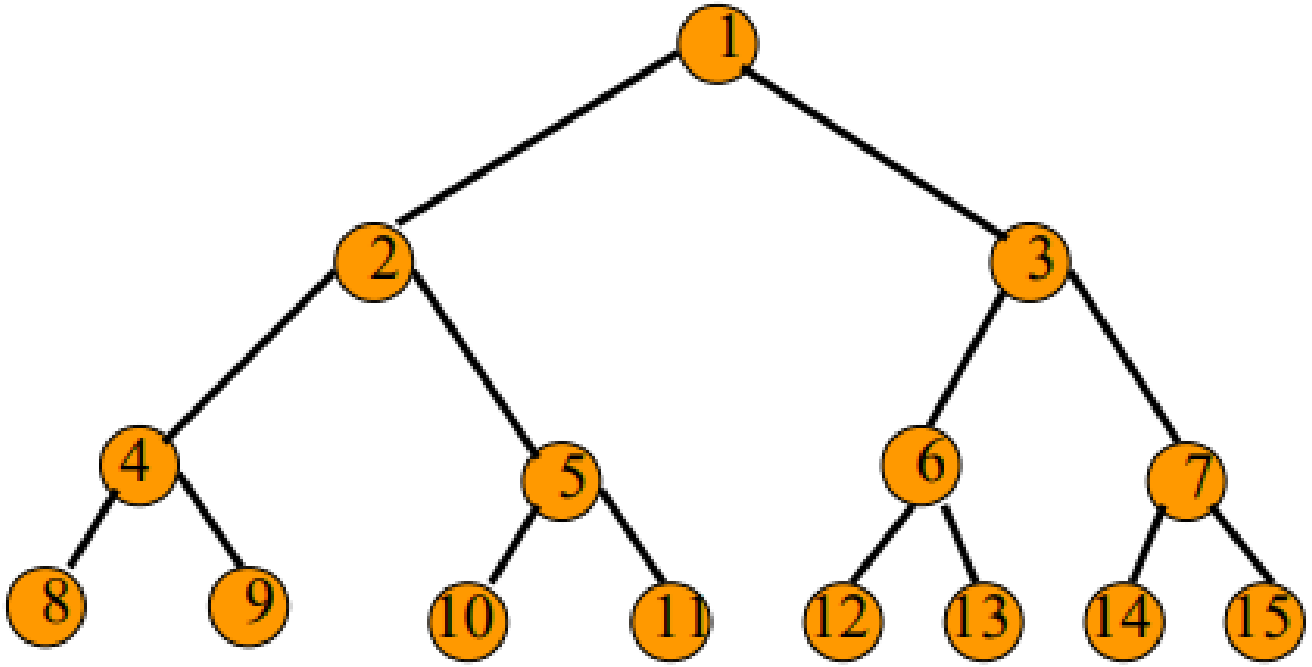
Height 4 full binary tree.

Numbering Nodes In A Full Binary Tree

- Number the nodes 1 through $2^h - 1$.
- Number by levels from top to bottom.
- Within a level number from left to right.

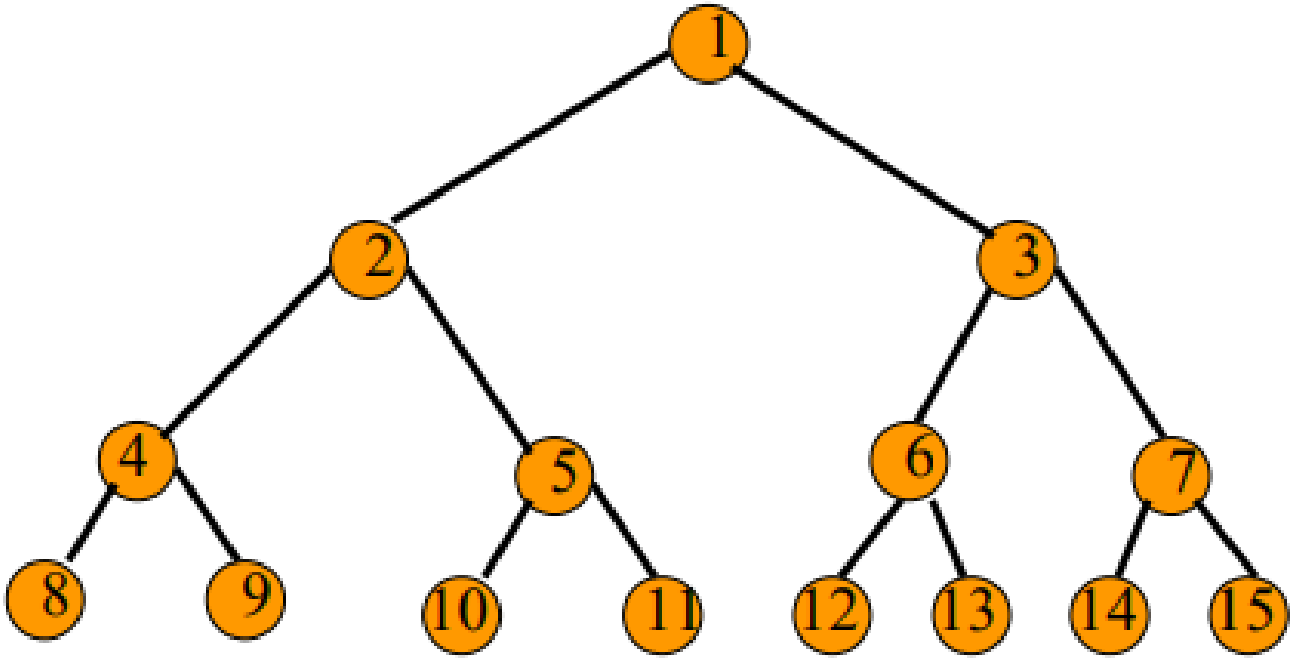


Node Number Properties



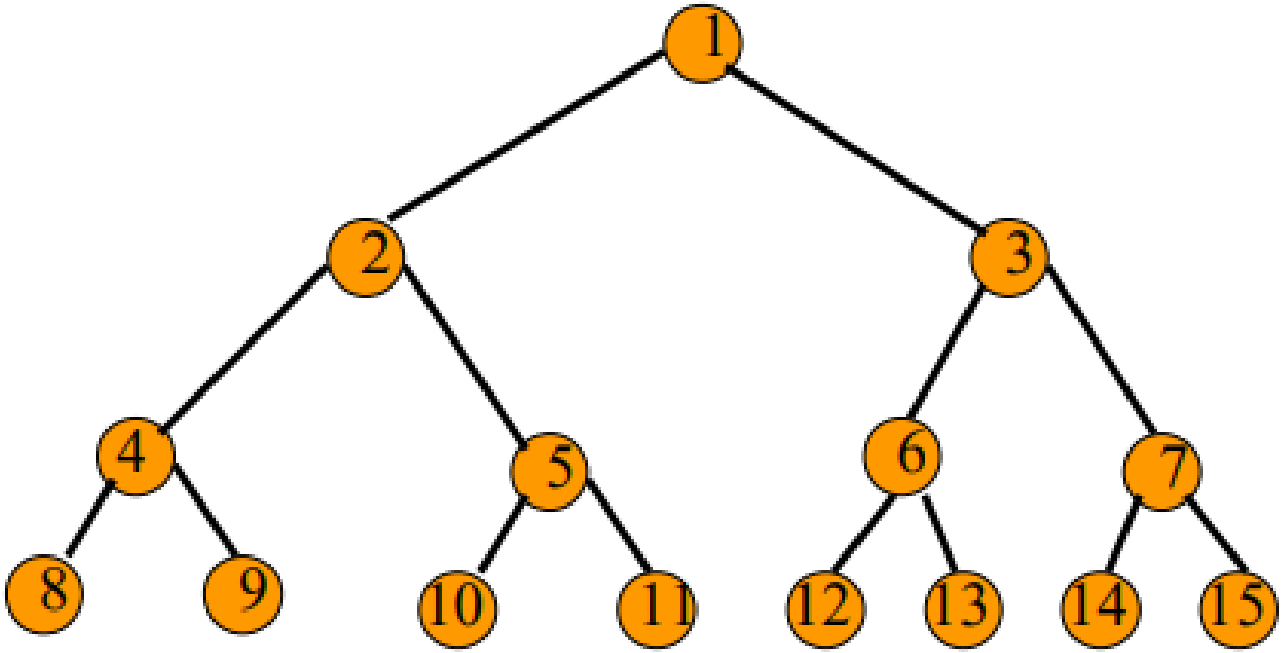
- Parent of node i is node i / 2, unless i = 1.
- Node 1 is the root and has no parent.

Node Number Properties



- Left child of node i is node 2i, unless 2i > n, where n is the number of nodes.
- If 2i > n, node i has no left child.

Node Number Properties

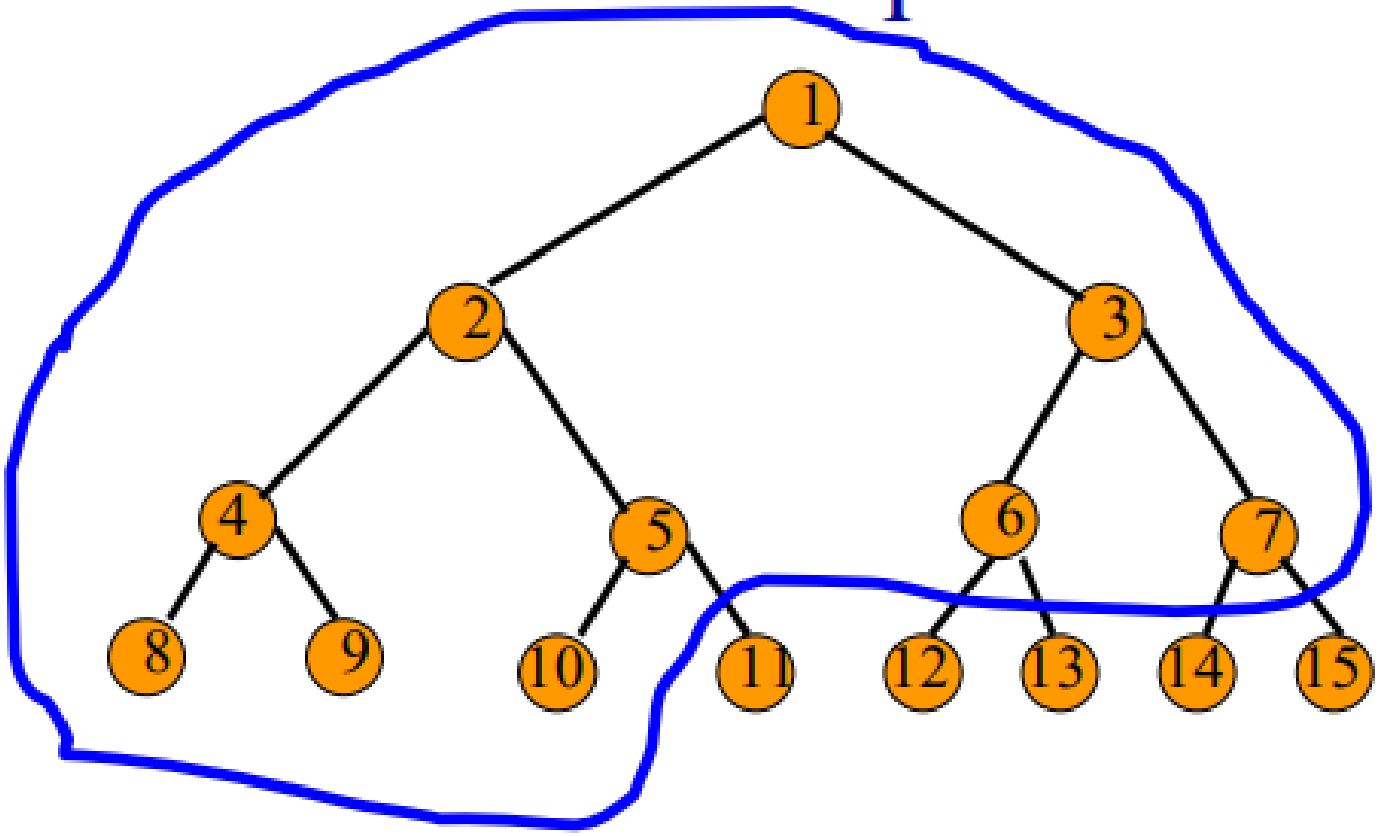


- Right child of node i is node 2i+1, unless 2i+1 > n, where n is the number of nodes.
- If 2i+1 > n, node i has no right child.

Complete Binary Tree With n Nodes

- Start with a full binary tree that has at least n nodes.
- Number the nodes as described earlier.
- The binary tree defined by the nodes numbered 1 through n is the unique n node complete binary tree.

Example

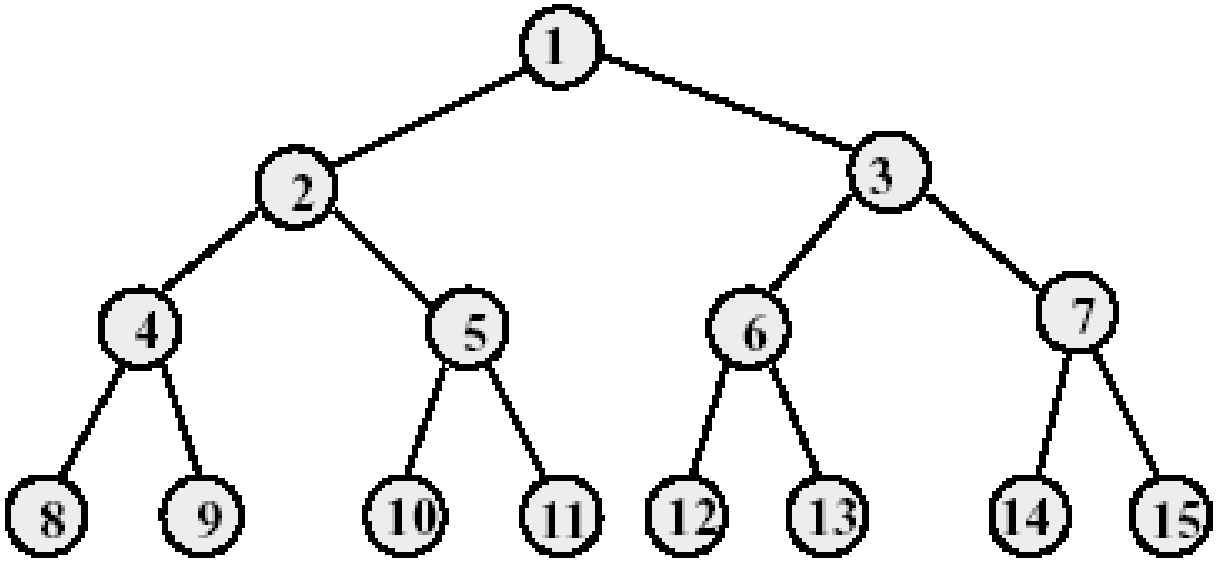


- Complete binary tree with 10 nodes.

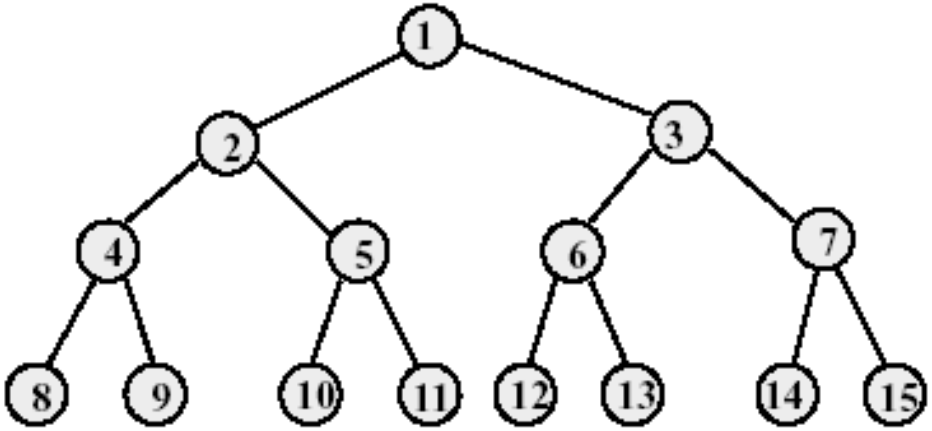
A **full binary tree** of height *h* has exactly ***2^h^*-1 nodes**.

Numbering the nodes in a full binary tree

- Number the nodes 1 through 2^h-1
- Number by levels from top to bottom
- Within a level, number from left to right

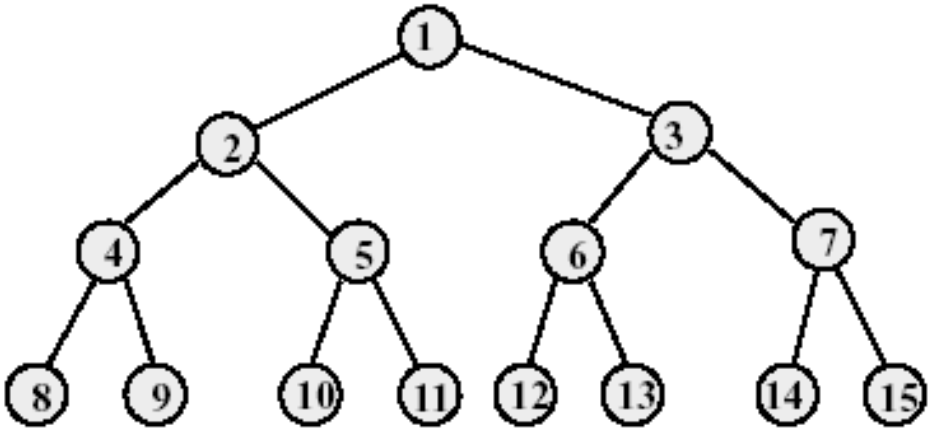


Node Number Property of Full Binary Tree



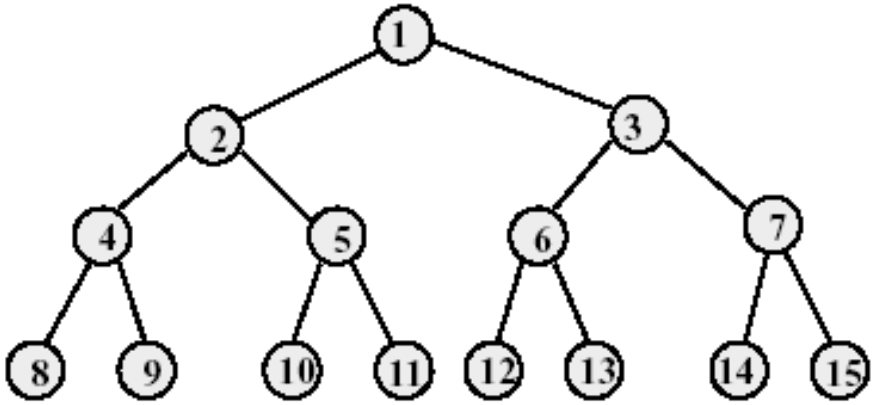
- Parent of node *i* is node $\lfloor (i/2) \rfloor$, unless i = 1
- Node 1 is the root and has no parent

Node Number Property of Full Binary Tree



- Left child of node *i* is node *2i*, unless *2i* > *n*, where *n* is the total number of nodes.
- If *2i* > *n*, node *i* has no left child.

Node Number Property of Full Binary Tree

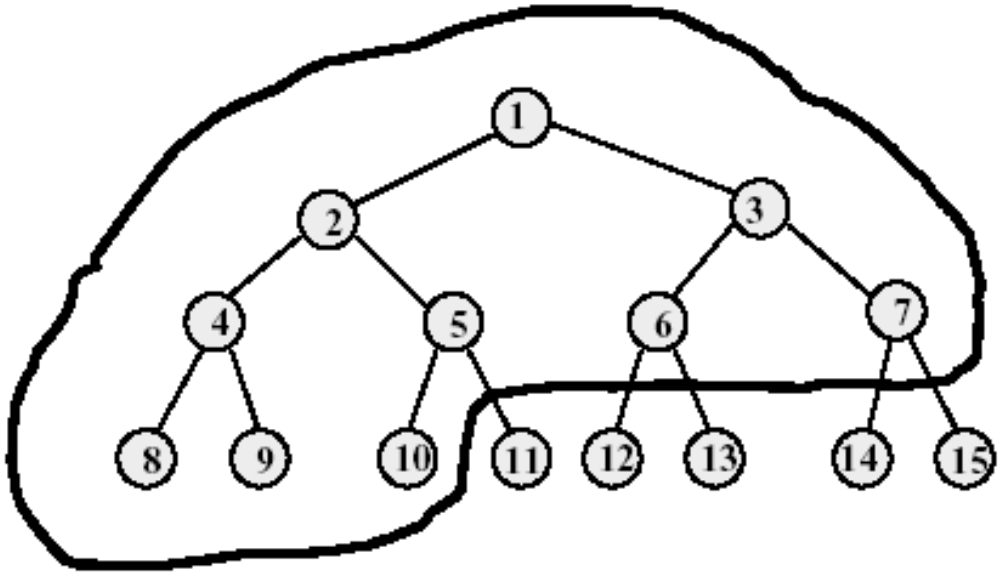


- Right child of node *i* is node *2i*+1, unless *2i*+1 > *n*,
  where n is the total number of nodes.
- If *2i+1 > n*, node *i* has no right child.

Complete Binary Tree with N Nodes

- Start with a full binary tree that has at least *n* nodes
- Number the nodes as described earlier.
- The binary tree defined by the nodes numbered 1 through *n* is the *n*-node complete binary tree.
- A full binary tree is a special case of a complete binary tree

Example of Complete Binary Tree

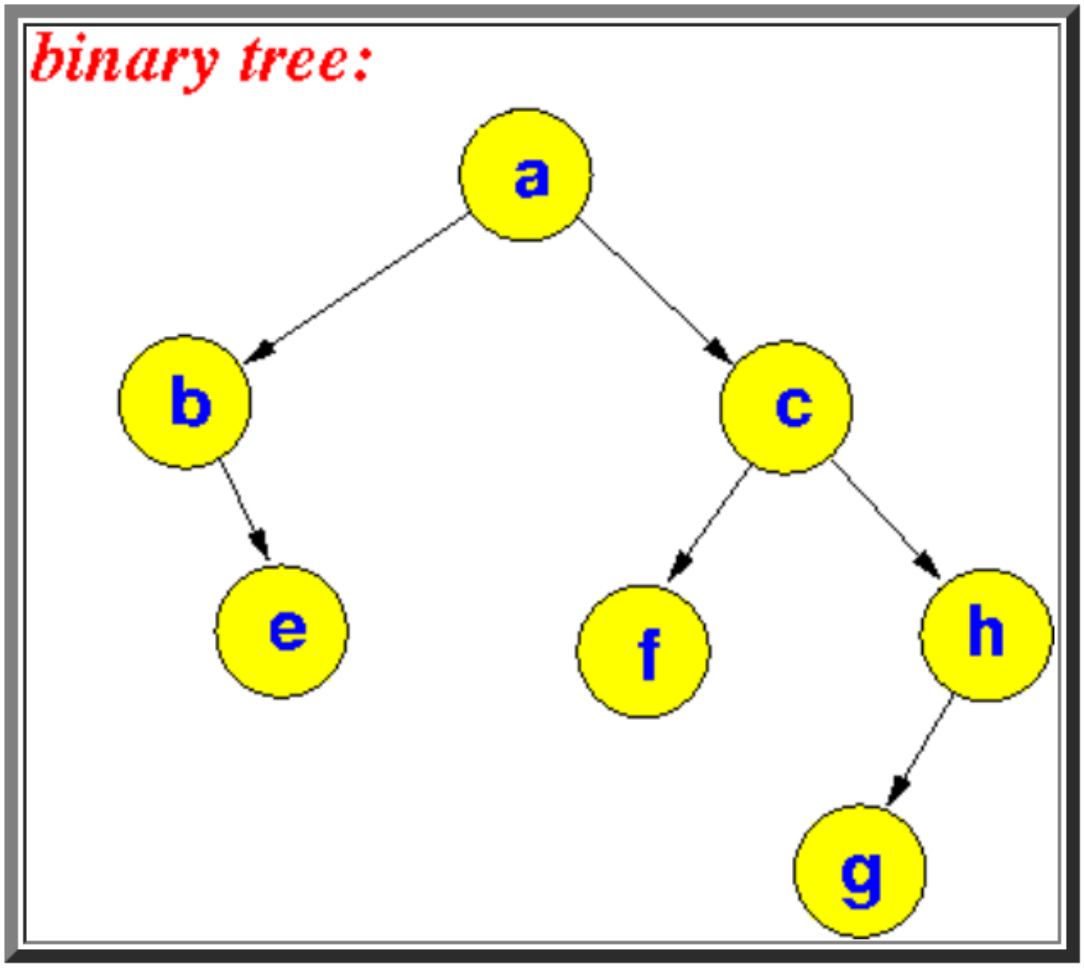


- Complete binary tree with 10 nodes.
- Same node number properties (as in full binary tree) also hold here.

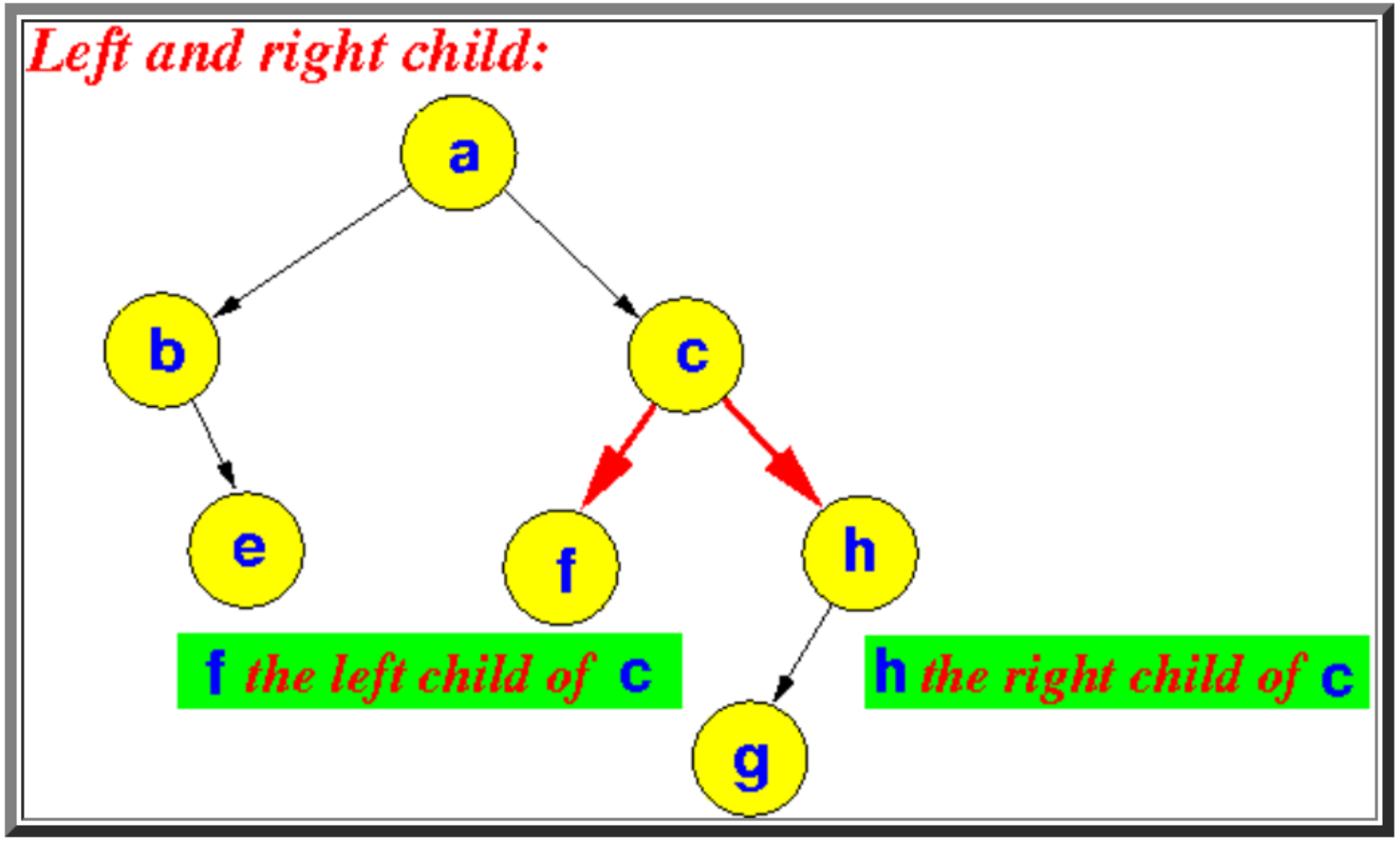
- ***Binary* tree**:

  - **Binary tree** = a **tree** where ***each* node** has ***at most* 2 children nodes**

Example:



- **Left and right child**
  - Because **each node** has *at most* **2** children nodes, we can **label** the children **distinctly** as **left** and **right**:



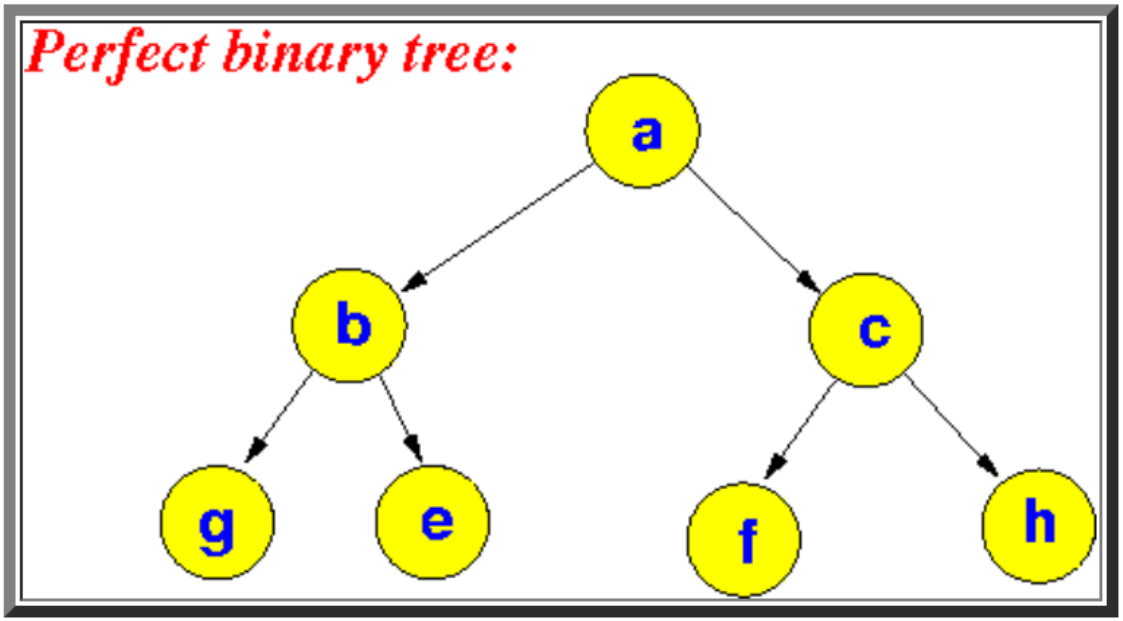
Note:

- Some nodes (e.g. *h*) have *only* a *left* child node
- Some nodes (e.g. *b*) have *only* a *right* child node

- **Perfect binary tree**
  - **Perfect** binary tree

> - **Perfect binary tree** = a **binary tree** where **each level** contains the ***maximum*** **number of nodes**
>
>   I.e., **every level** is **completely** ***full*** of nodes

Example:



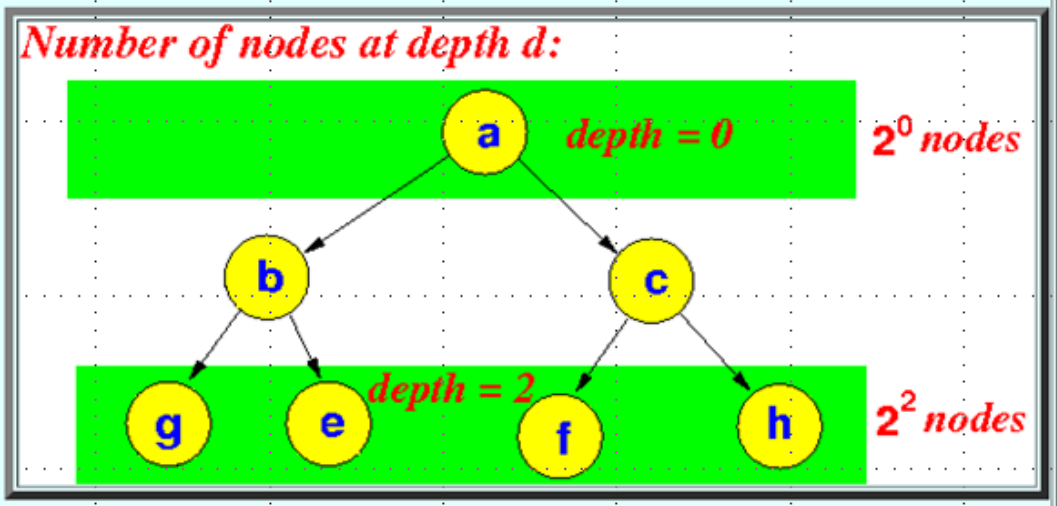
- **Some properties of the *perfect* binary tree**
  - **Property 1:**
    - The **number of nodes** at **depth** d in a ***perfect*** **binary tree** = 2^d

Proof:

- There is *only* 1 node (= the root node) at depth *0*:

$2^0 = 1$

- In a perfect binary tree, every node has 2 children nodes



So:

Depth d	# nodes at depth d	# of child nodes
0	1 = 2^0	2 (each node has 2 children)
1	2 = 2^1	4 (each node has 2 children)
2	4 = 2^0	8 (each node has 2 children)
...		

I.e.:

- The number of nodes doubles every time the depth increases by 1 !

Therefore:

nodes at depth d = 2^d

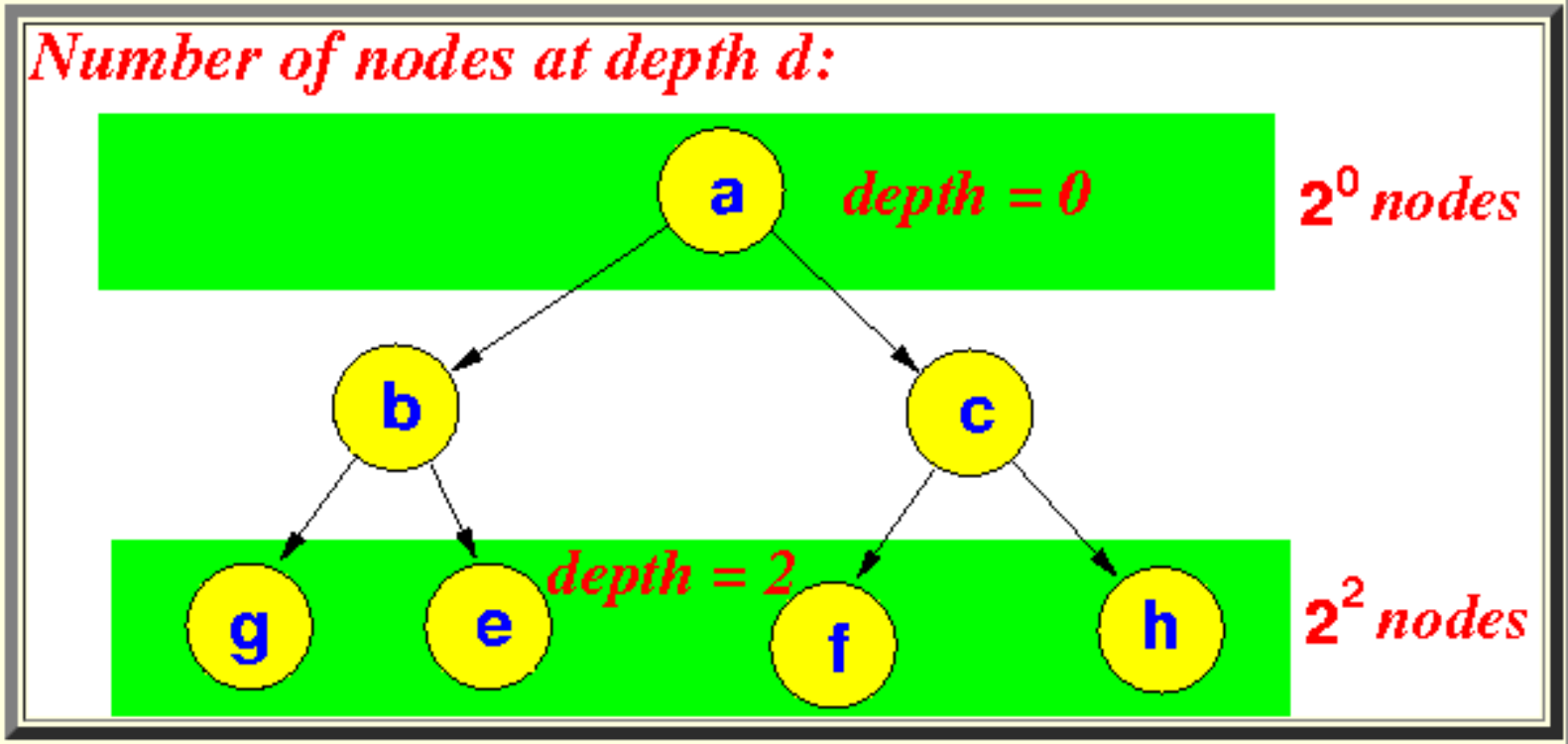
Property 2:

- A *perfect* binary tree of height *h* has:
  - $2^{h+1} - 1$ nodes

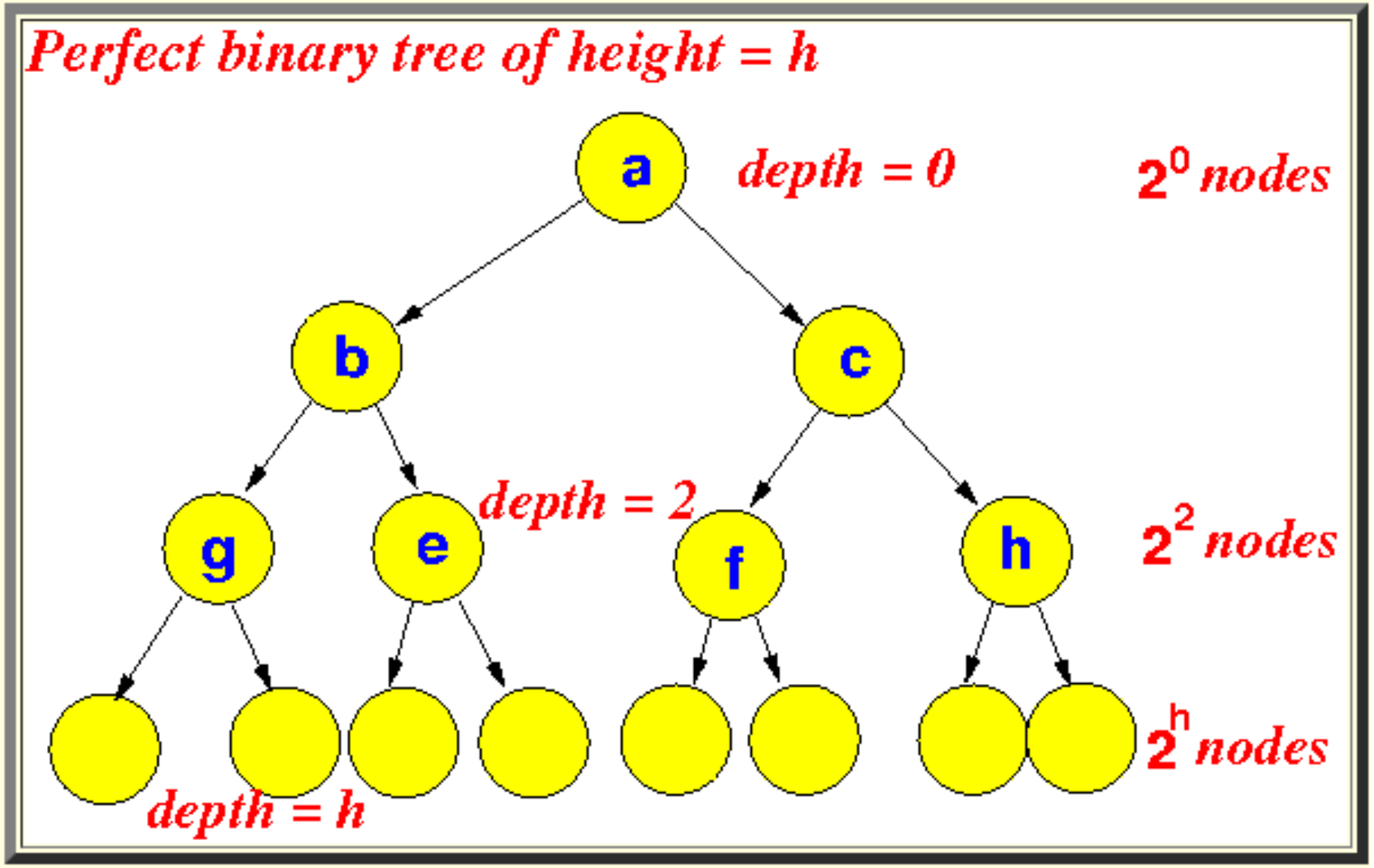
- **Previously,** we have shown that:

  - # nodes at *depth* d = 2^d

See:



- So the **total number** of **nodes** in a **perfect binary tree** of **height** *h*:



nodes = $2^0 + 2^1 + \ldots 2^h = 2^{h+1} - 1$

- **Proof:**

$$
\begin{aligned}
S &= 1 + 2 + 2^2 + 2^3 + \ldots + 2^h \\
2xS &= \quad 2 + 2^2 + 2^3 + \ldots + 2^h + 2^{h+1} \quad \text{- (subtract)} \\
\hline
2xS - S &= 2^{h+1} - 1 \\
\Longleftrightarrow \quad S &= 2^{h+1} - 1
\end{aligned}
$$

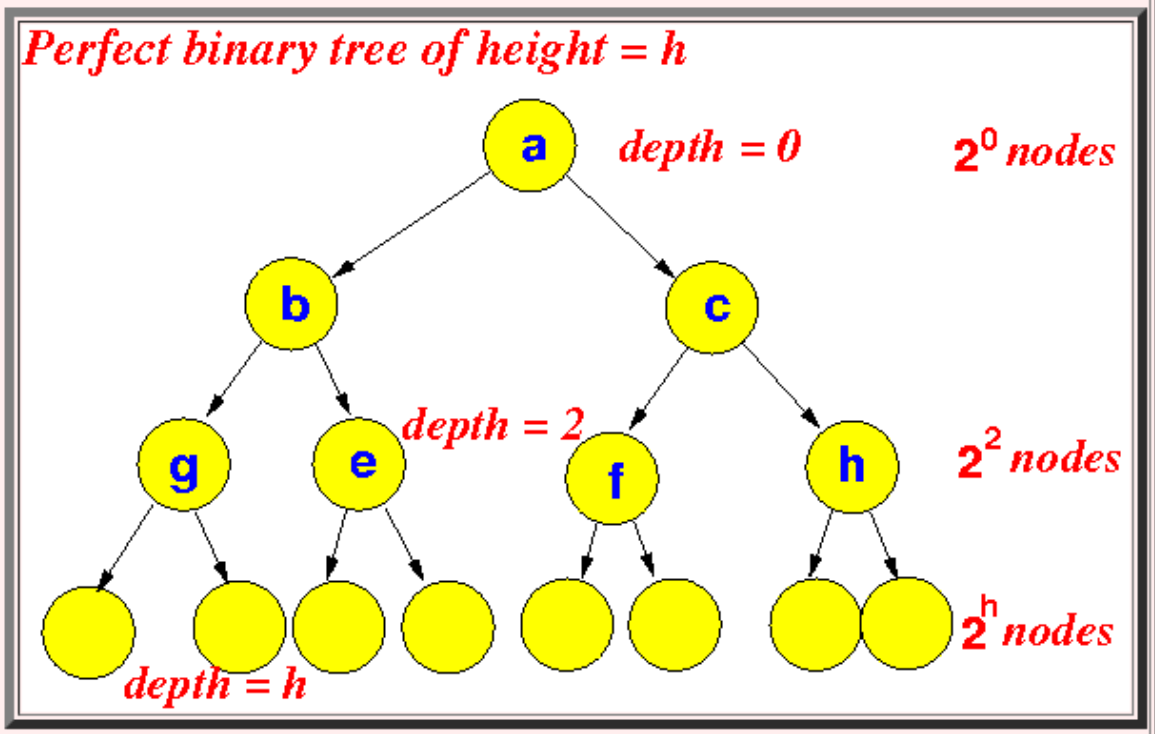
- **Property 3:**

  - **Number** of ***leaf*** **nodes** in a **perfect binary tree** of **height** $h = 2^h$

Proof:

- # nodes at **depth** d in a **perfect binary tree** = 2^d

- **All** the ***leaf*** **nodes** in a **perfect binary tree** of **height** h has a depth equal to h:



- # nodes at **depth** h in a **perfect binary tree** = 2^h

Therefore:

  - **Number** of ***leaf*** **nodes** in a **perfect binary tree** of **height** $h = 2^h$

- **Property 4:**

  - **Number** of ***internal*** **nodes** in a **perfect binary tree** of **height** $h = 2^h - 1$

Proof:

- # nodes in a **perfect binary tree** of **height** $h = 2^{h+1} - 1$ (see **Property 2**)

- # ***leaf*** nodes in a **perfect binary tree** of **height** $h = 2^h$ (see **Property 3**)

- The **other nodes** are ***internal*** **nodes** (i.e., with at least 1 child node).

  So:

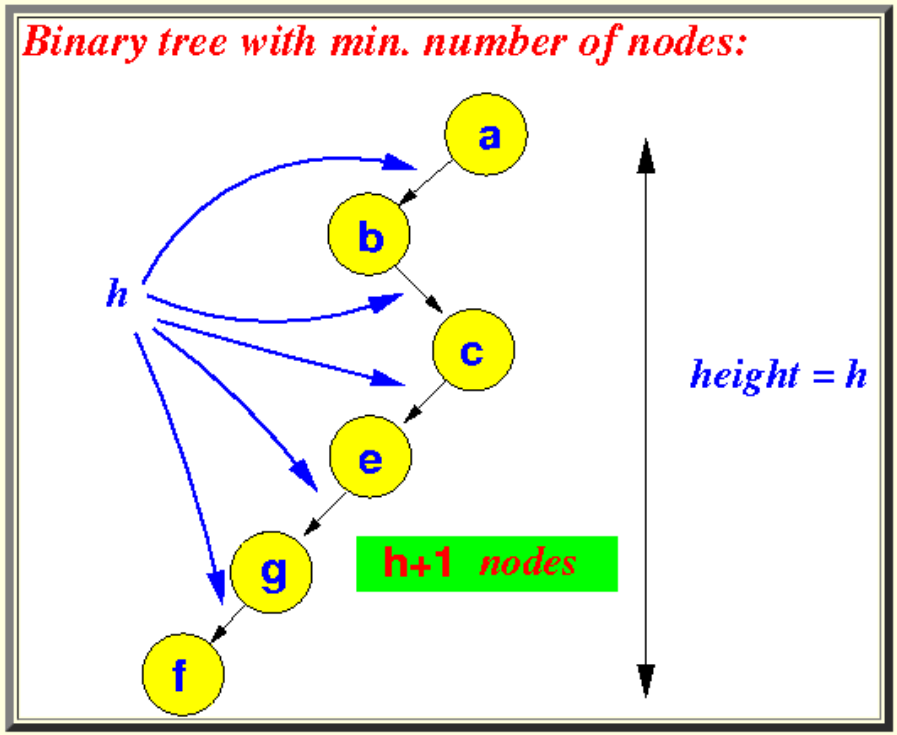
  - # **internal** nodes in a **perfect binary tree** of **height** $h = (2^{h+1} - 1) - 2^h = 2^h - 1$

- **Minimum and maximum number of nodes in a binary tree of height *h***
  - **Fact:**

    - The ***minimum* number of nodes** in a **binary tree** of ***height* h** = ***h*** + 1

    Proof:

    - The **binary tree** of ***height* h** with the ***minimum* number of nodes** is a tree where **each node** has ***one* child**:



    - Because the ***height* = h**, the are ***h* edges**

    - ***h* edges** connects ***h+1* nodes**

      - Therefore, the ***minimum* number of nodes** in a **binary tree** of ***height* h** = ***h*** + 1

- **Fact:**

  - The ***maximum*** **number of nodes** in a **binary tree** of ***height*** **h** = $2^{h+1} - 1$

Proof:

- The ***perfect*** **binary tree** has the **maximum number of nodes**

- We have already shown that:

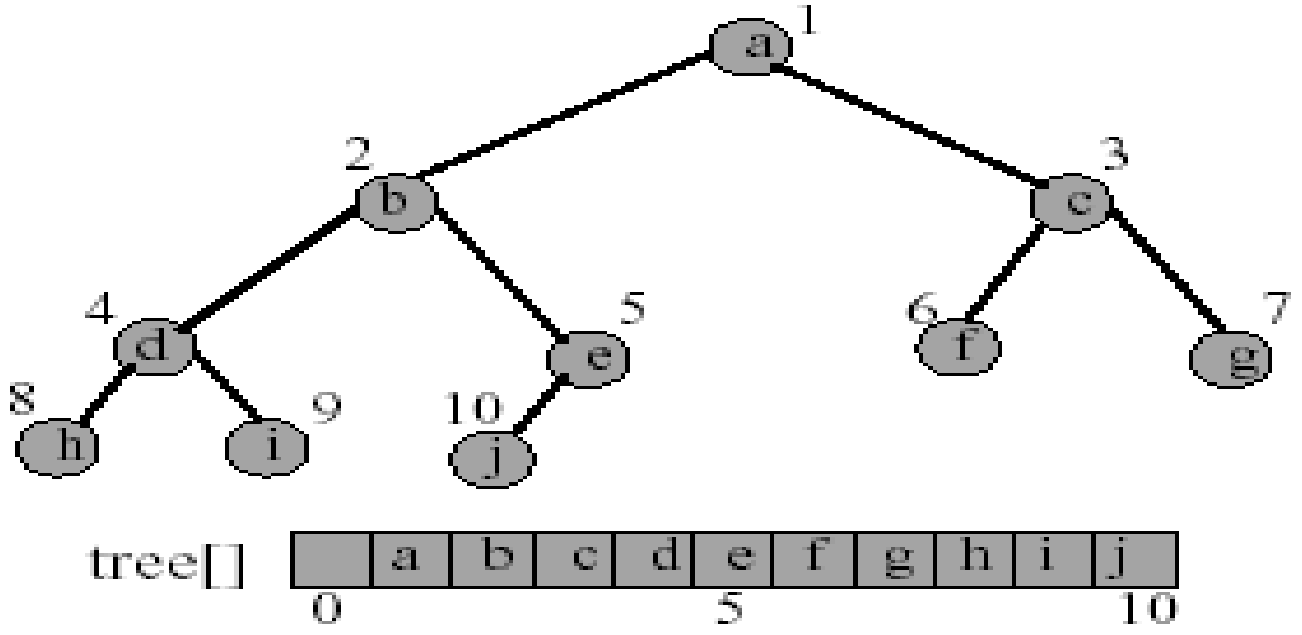
  - # nodes in a ***perfect*** **binary tree** = $2^{h+1} - 1$

Binary Tree Representation

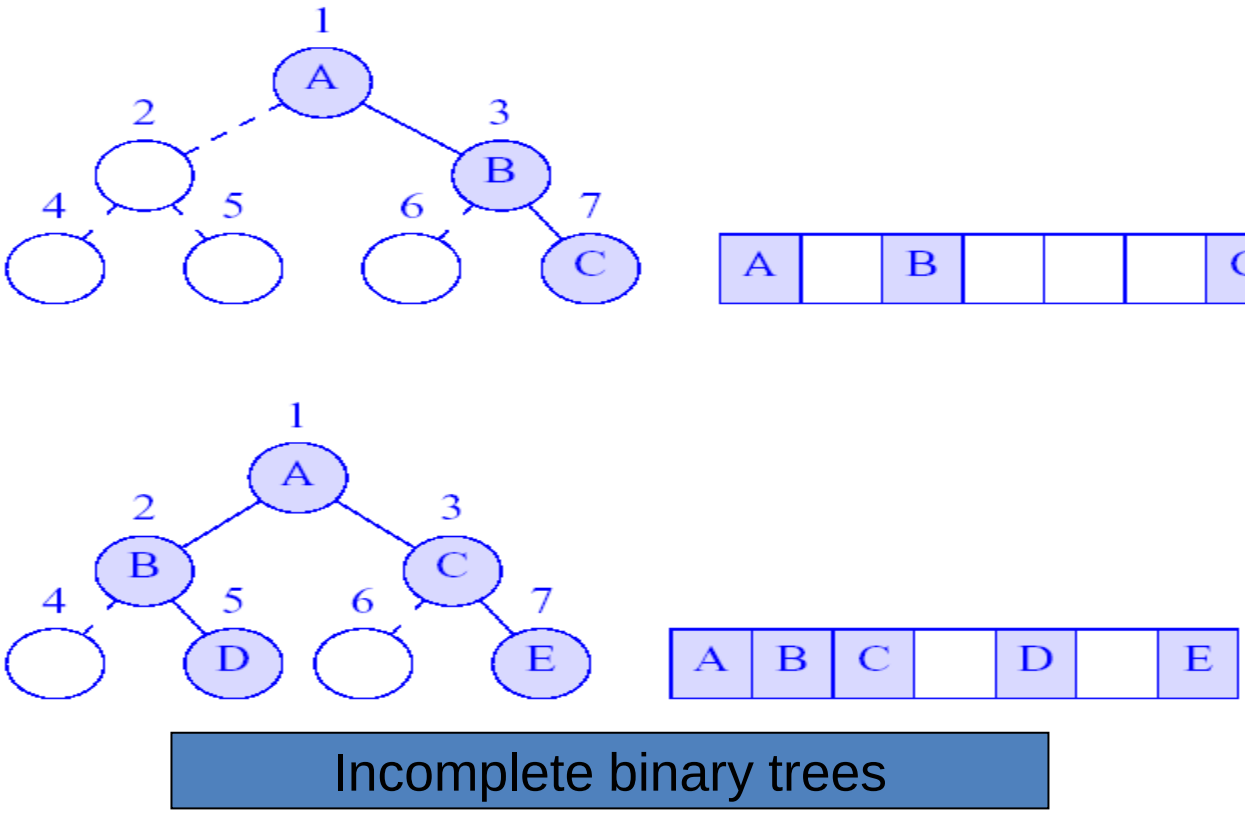
- Array representation
- Linked representation

Array Representation of Binary Tree

- The binary tree is represented in an array by storing each element at the array position corresponding to the number assigned to it.



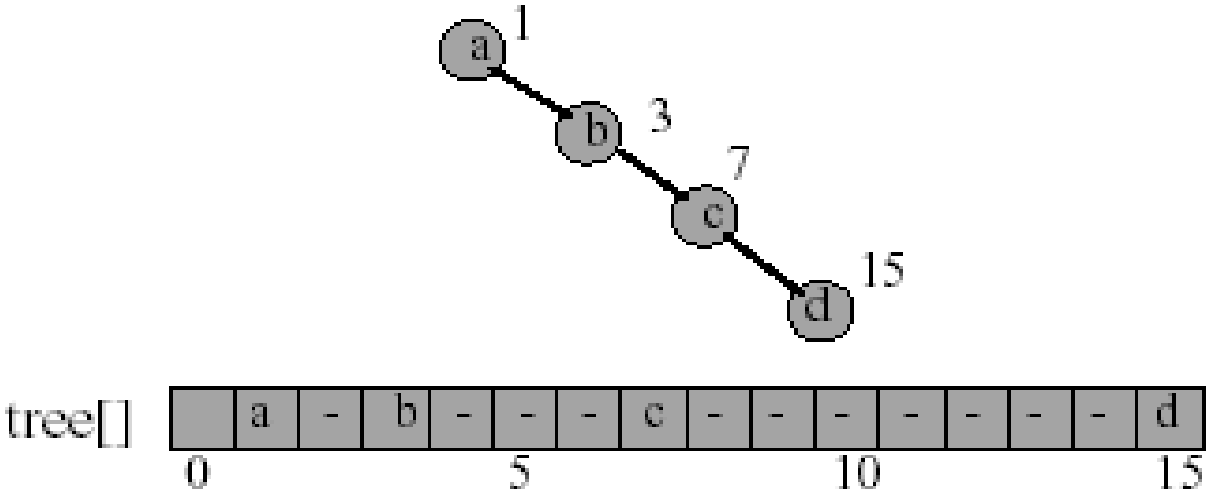
Incomplete Binary Trees



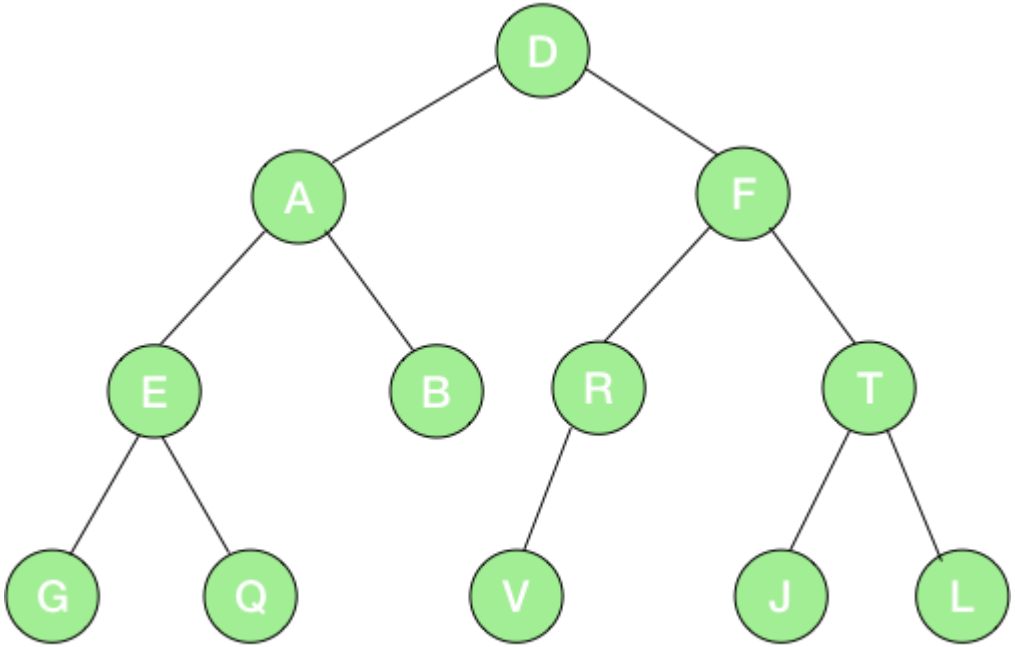
Incomplete binary trees

- Complete binary tree with some missing elements

Right-Skewed Binary Tree

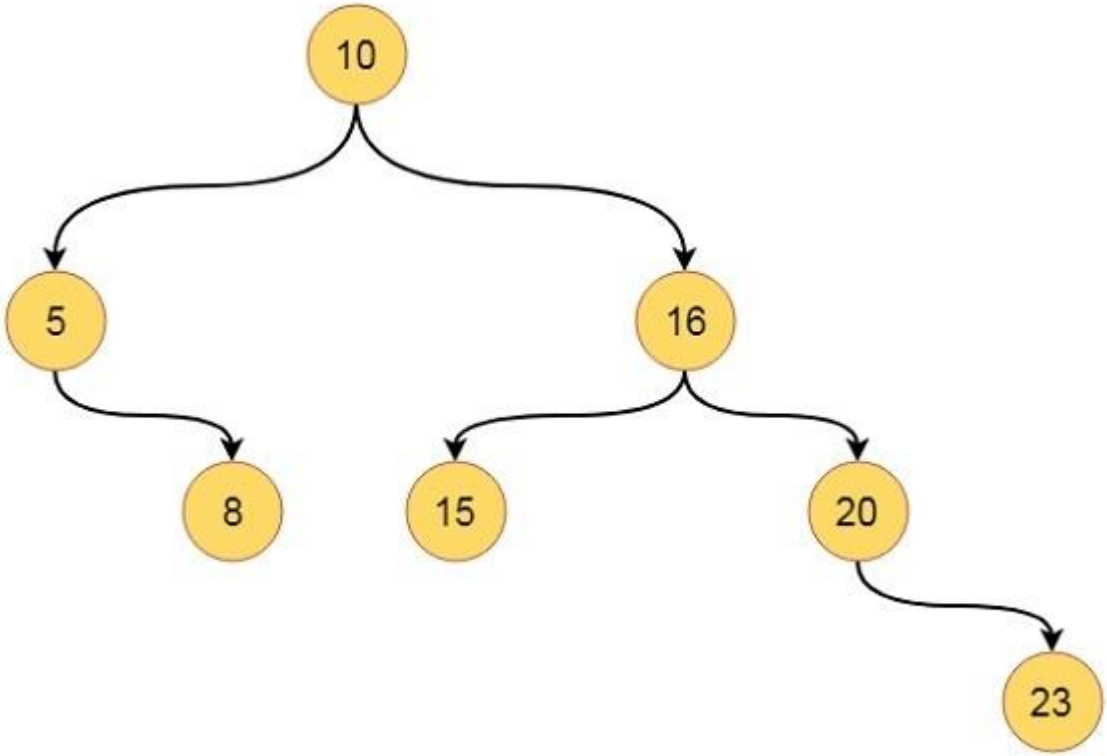


- An *n* node binary tree needs an array whose length is between ***n+1*** and $\boldsymbol{2^n}$.
- Right-skewed binary tree wastes the most space
- What about left-skewed binary tree?

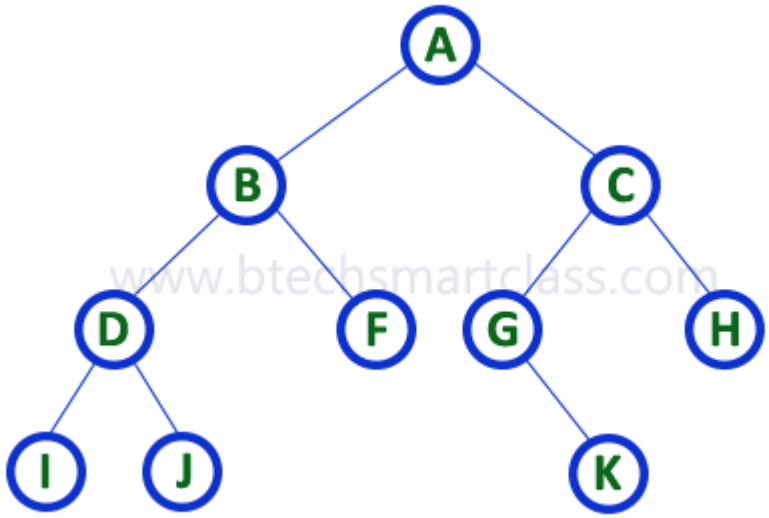


Binary Tree

\0	D	A	F	E	B	R	T	G	Q	\0	\0	V	\0	J	L



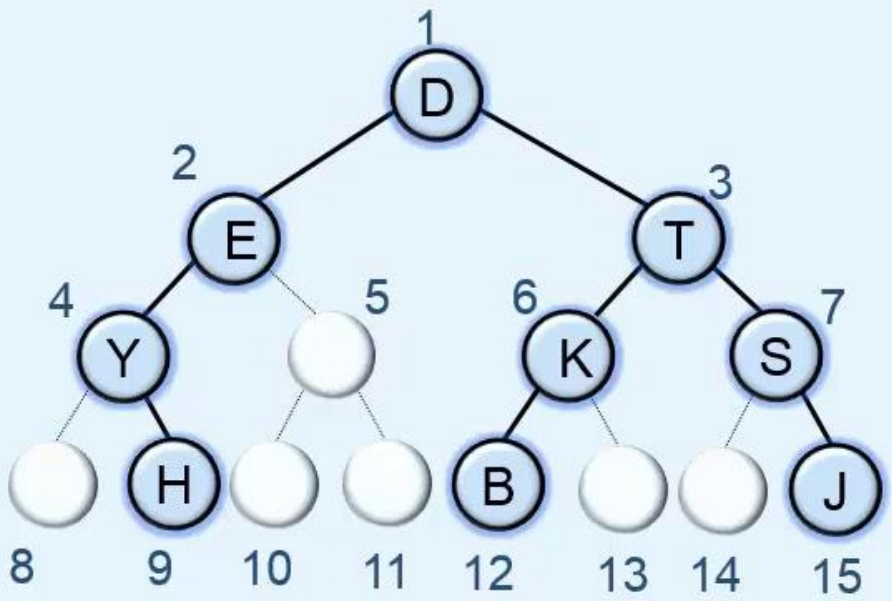
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	5	16	-	8	15	20	-	-	-	-	-	-	-	23



www.btechsmartclass.com

A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

Sequential Representation of Binary Trees

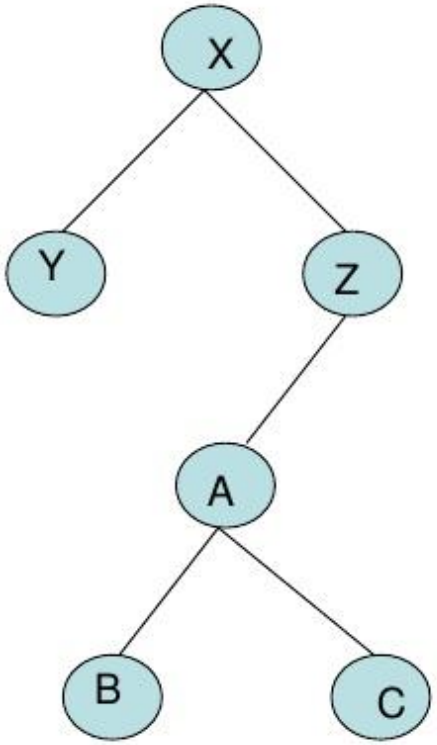


tree

	D	E	T	Y		K	S		H			B			J
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Representation of Binary Tree

- Array representation
  - The root of the tree is stored in position 0.
  - The node in position p, is the implicit father of nodes 2p+1 and 2p+2.
  - Left child is at 2p+1 and right at 2p+2.



0	1	2	3	4	5	6	7	8	9	10	11	12
X	Y	Z			A						B	C

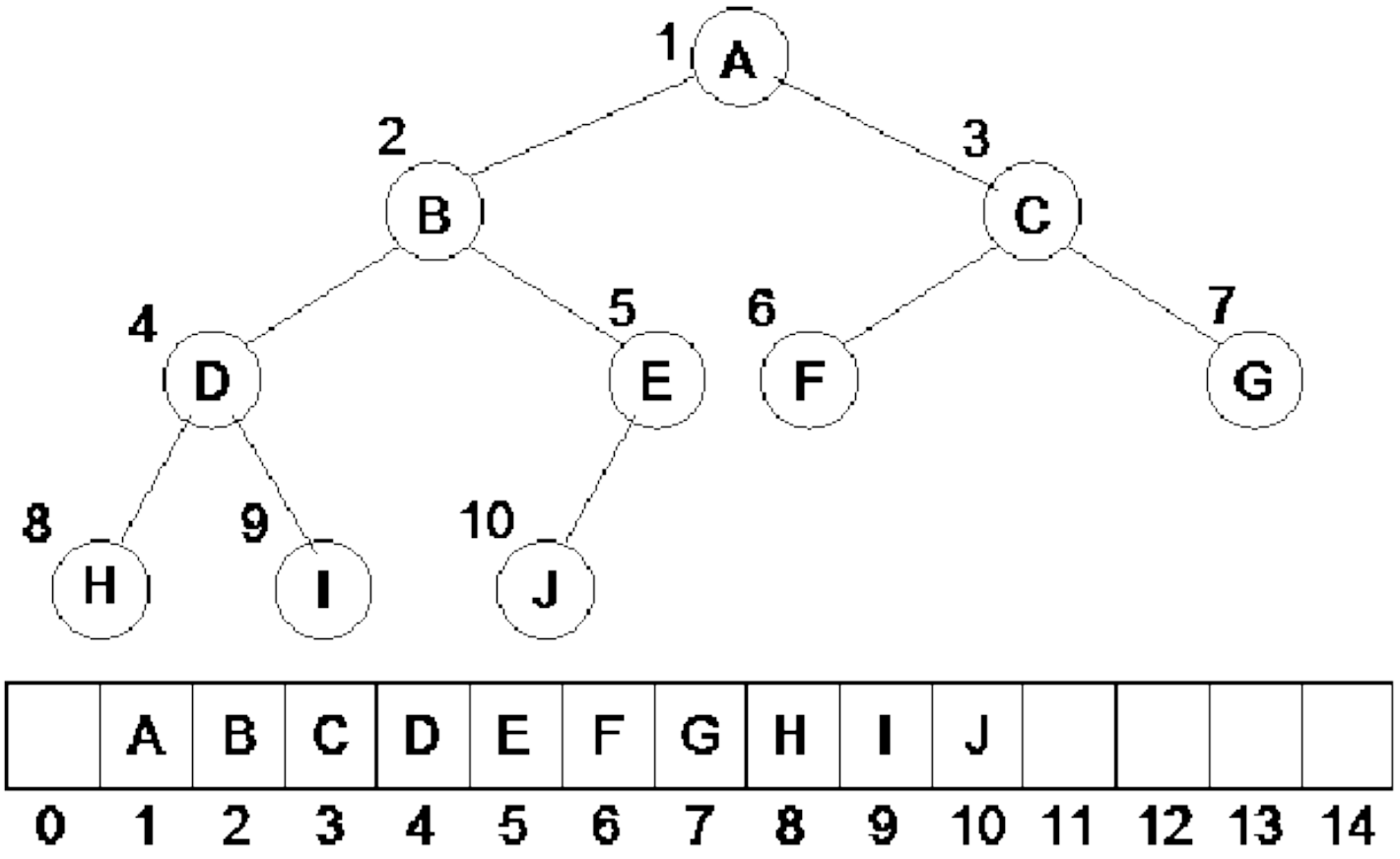
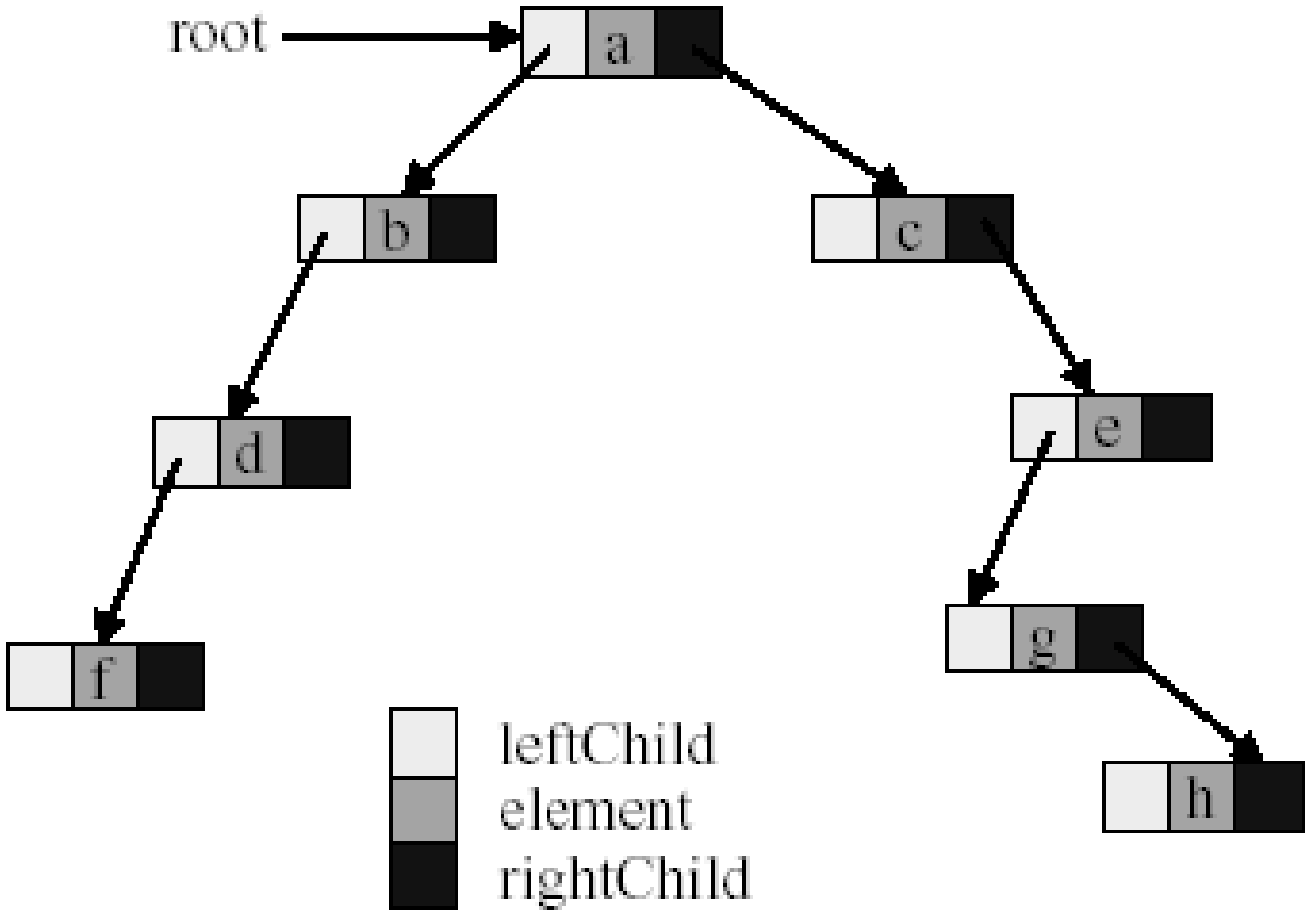


Figure 29.1: Complete Binary Tree

Linked Representation of Binary Tree

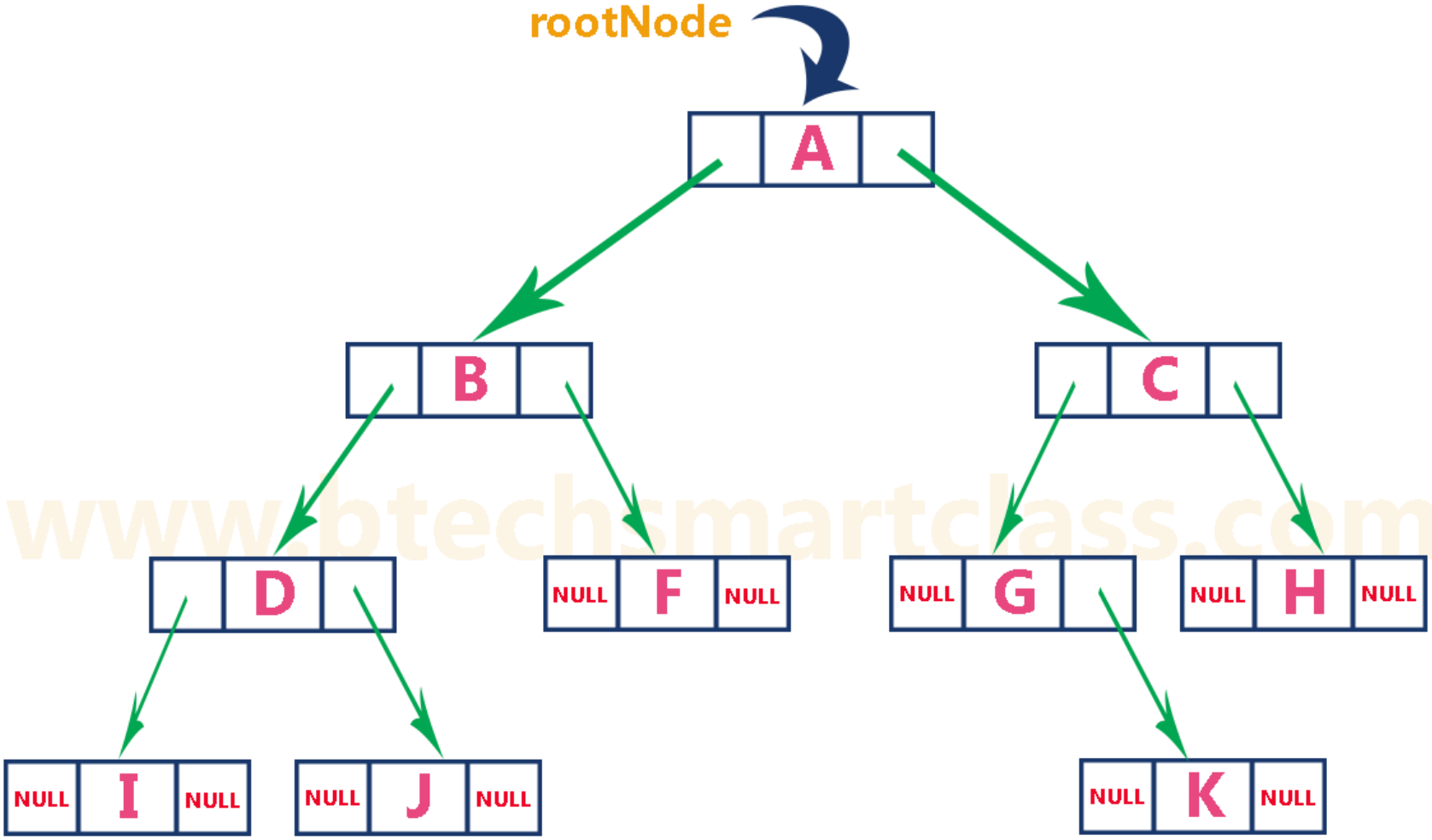
- The most **popular way to present a binary tree**
- Each element is represented by a **node** that has two link fields (leftChild and rightChild) plus an element field
- Each binary tree node is represented as an object whose data type is **binaryTreeNode**
- The space required by an *n* node binary tree is ***n* * sizeof(binaryTreeNode)**

Linked Representation of Binary Tree

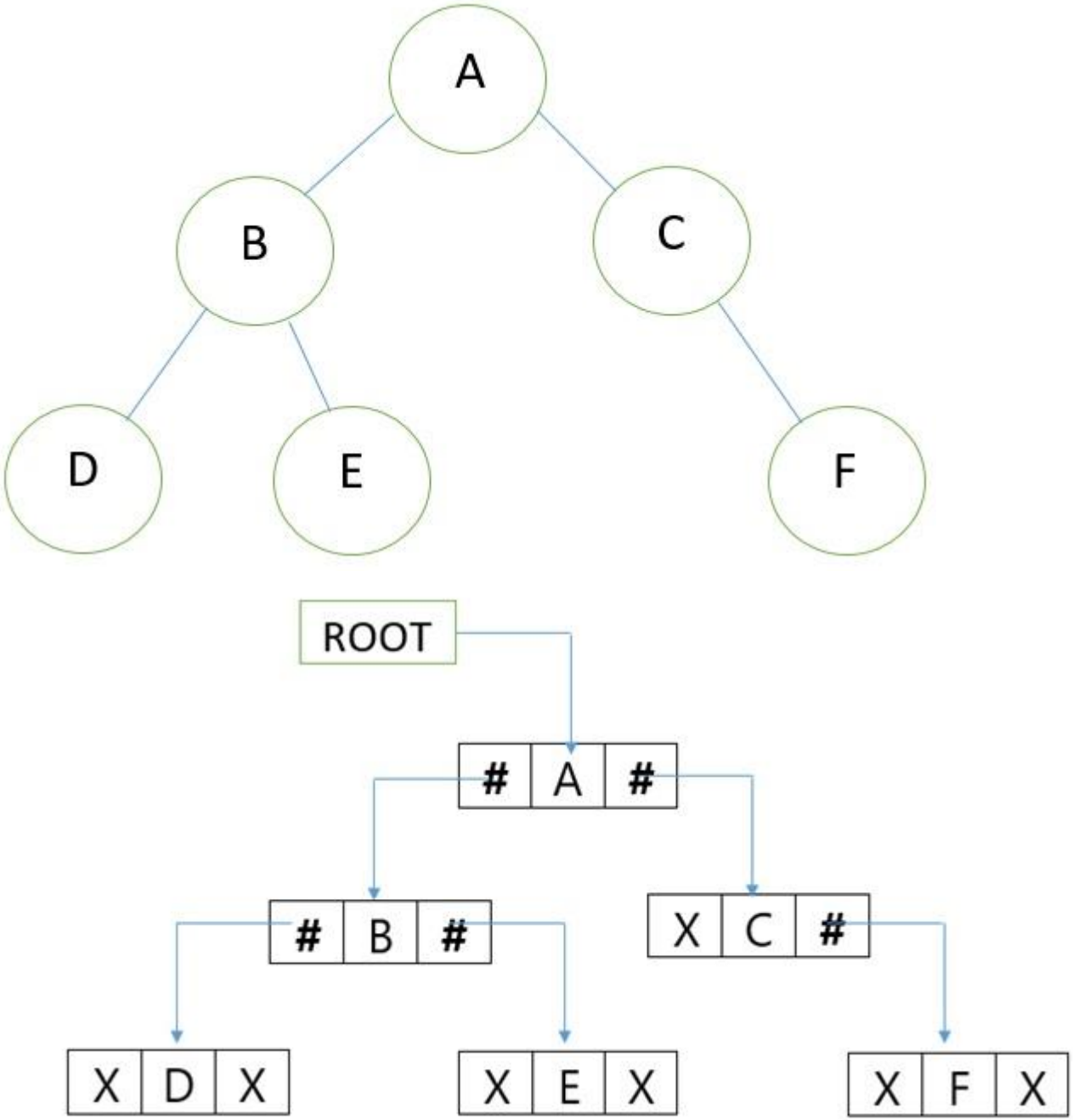


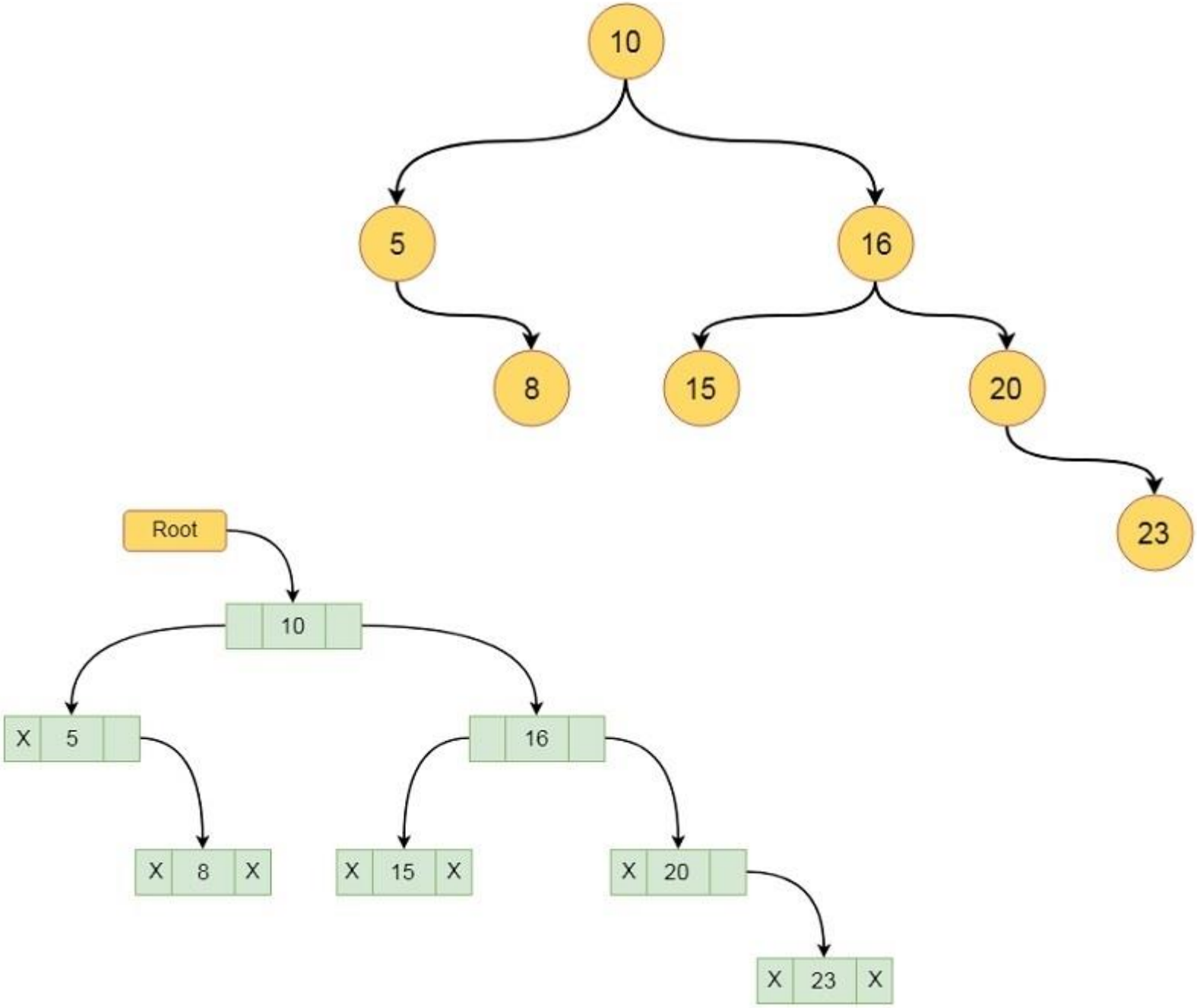
Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address. In this linked list representation, a node has the following structure…

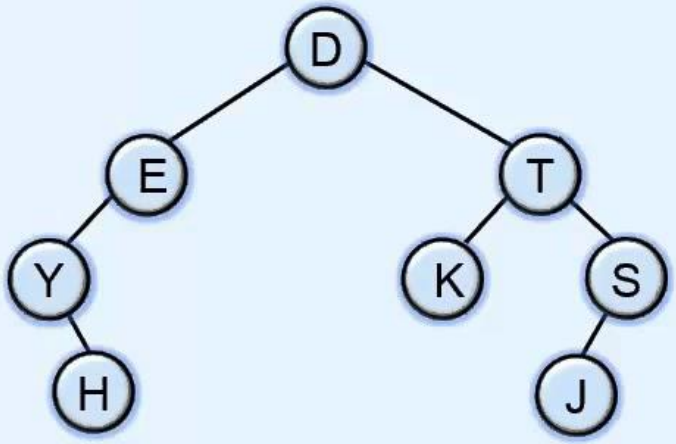


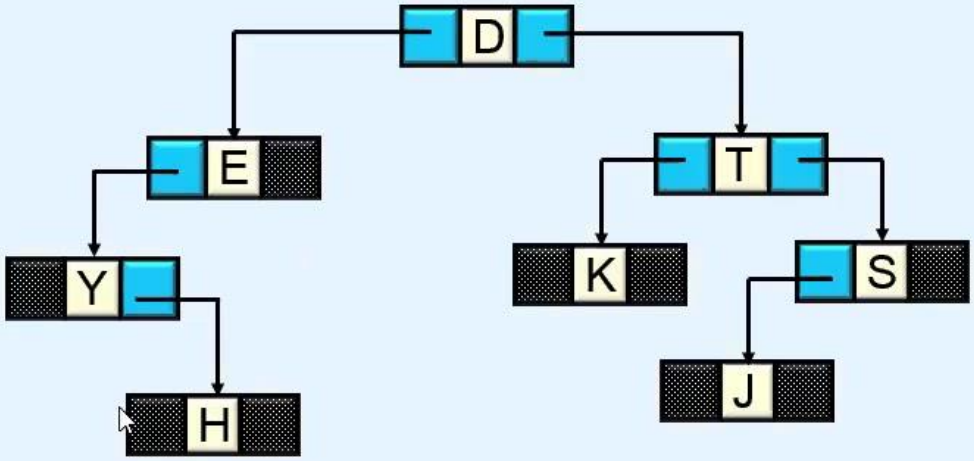
www.btechsmartclass.com

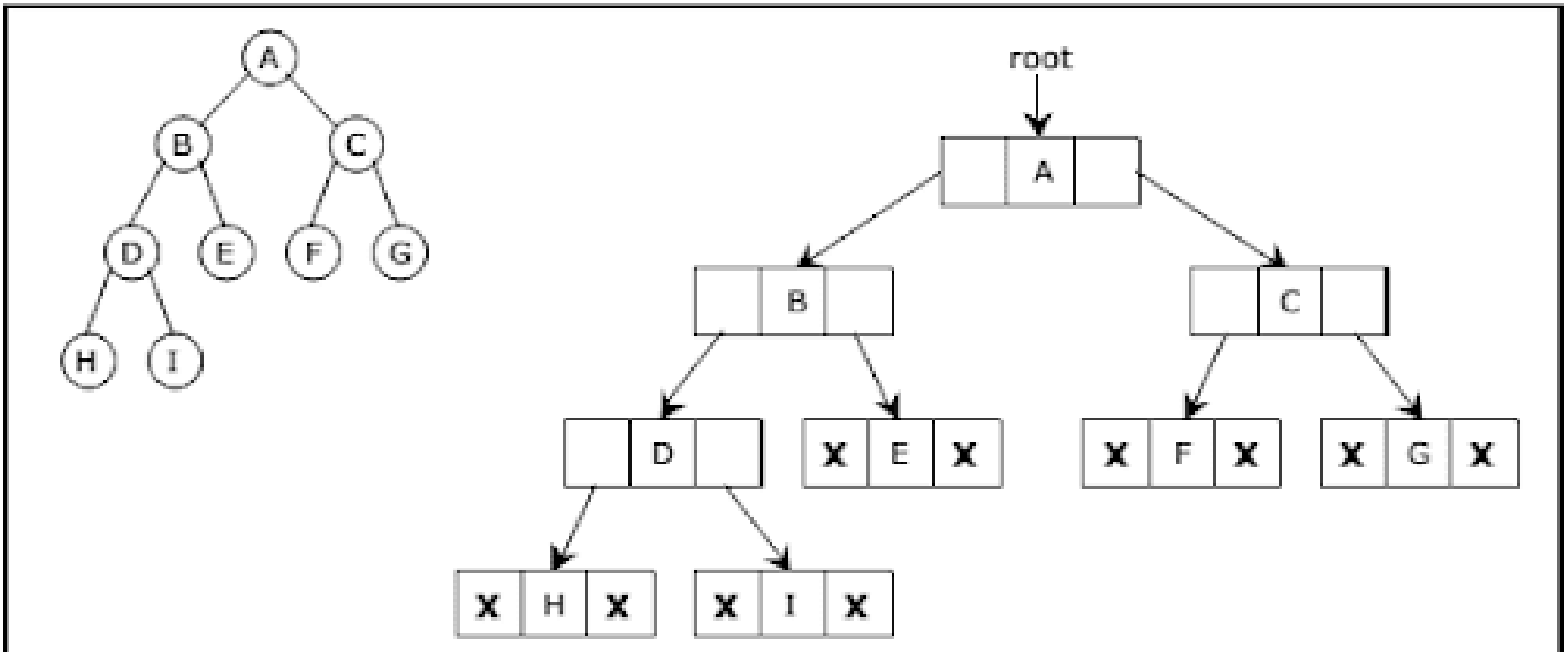

A
B
C
D
E
F
ROOT
A
B
X C #
X D X
X E X
X F X


10
5
16
8
15
20
23
Root
10
X
5
16
X
8
X
X
15
X
X
20
X
23
X

Linked Representation of Binary Trees



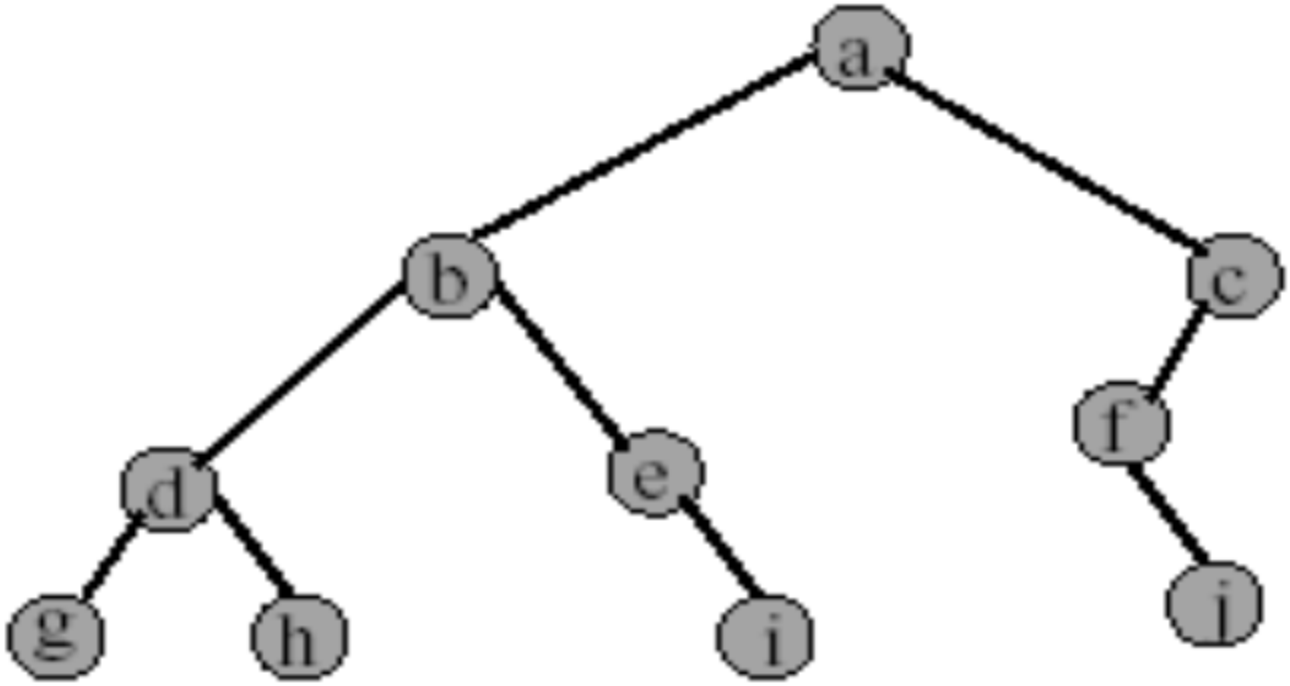



A
B
C
D
E
F
G
H
I
root
A
B
C
D
X
E
X
X
F
X
X
G
X
X
H
X
X
I
X

Binary Tree Traversal Methods

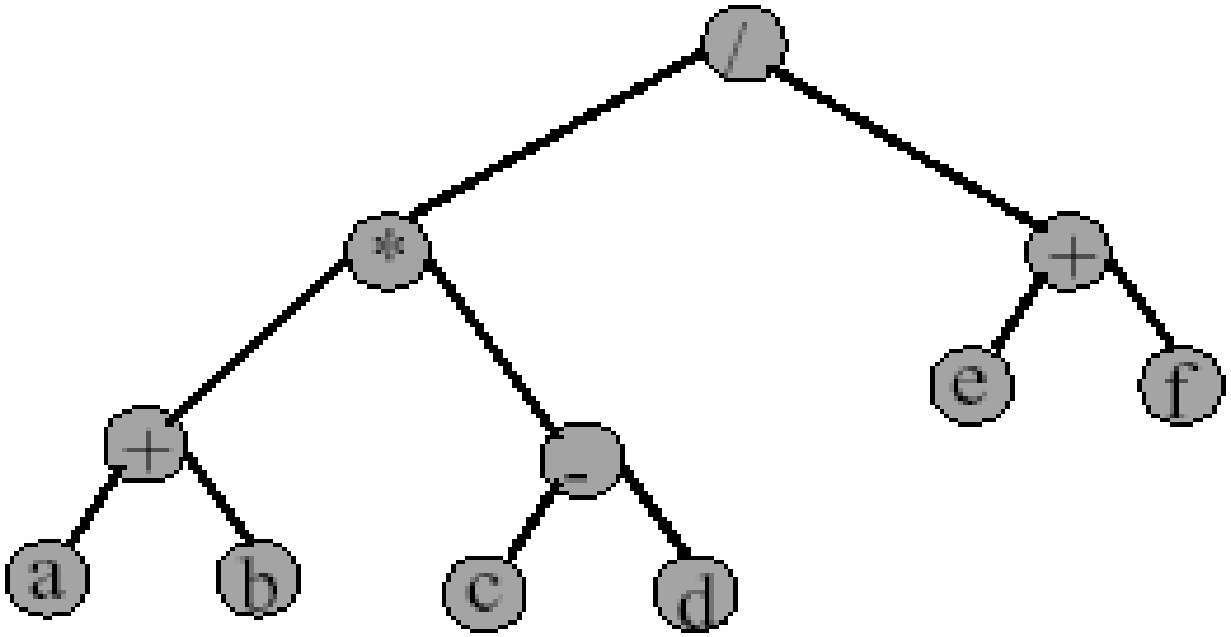
- **Preorder**
  - The root of the subtree is processed first before going into the left then right subtree (root, left, right).
- **Inorder**
  - After the complete processing of the left subtree the root is processed followed by the processing of the complete right subtree (left, root, right).
- **Postorder**
  - The root is processed only after the complete processing of the left and right subtree (left, right, root).
- **Level order**
  - The tree is processed by levels. So first all nodes on level i are processed from left to right before the first node of level i+1 is visited

Preorder Example



a b d g h e i c f j

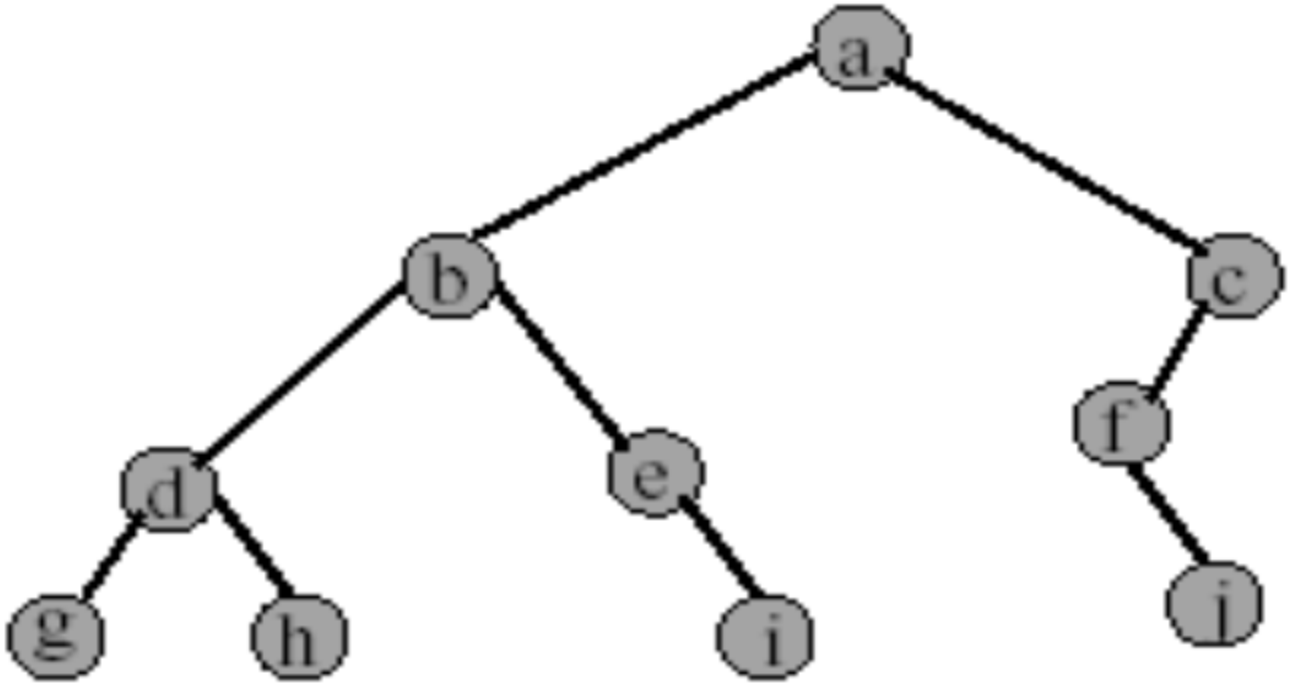
Preorder of Expression Tree



/ * + a b - c d + e f

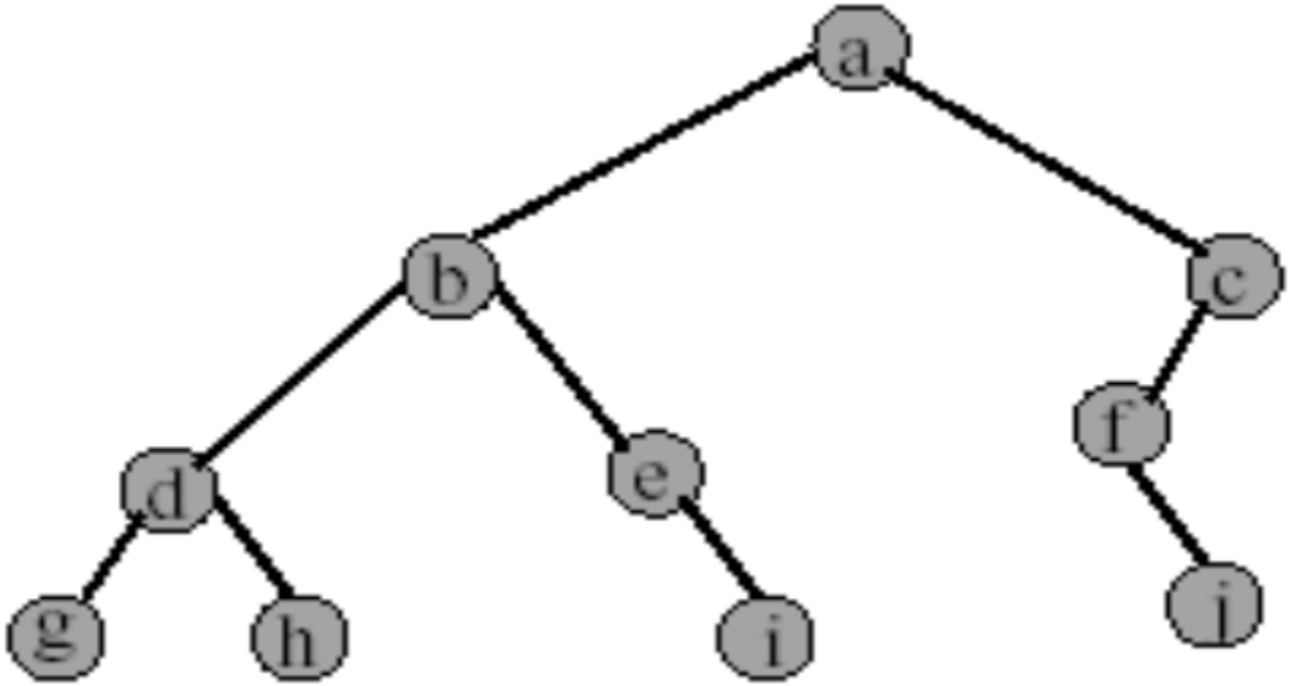
Gives prefix form of expression.

Inorder Example



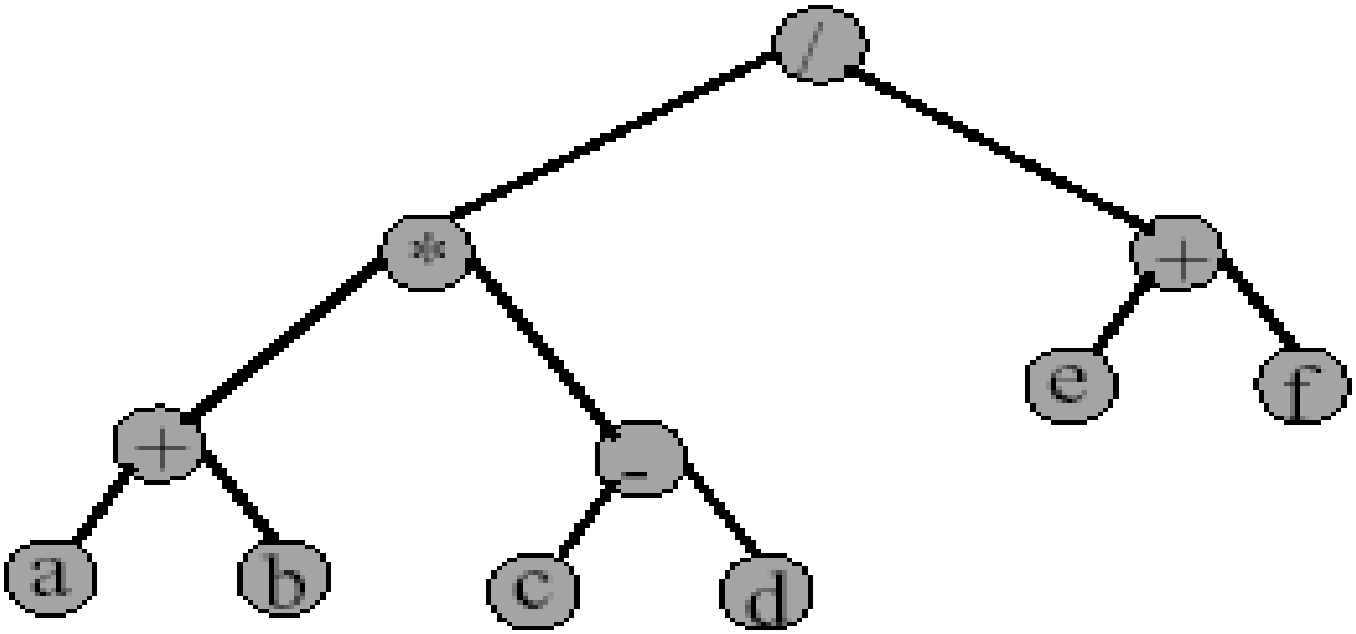
g d h b e i a f j c

Postorder Example



g h d i e b j f c a

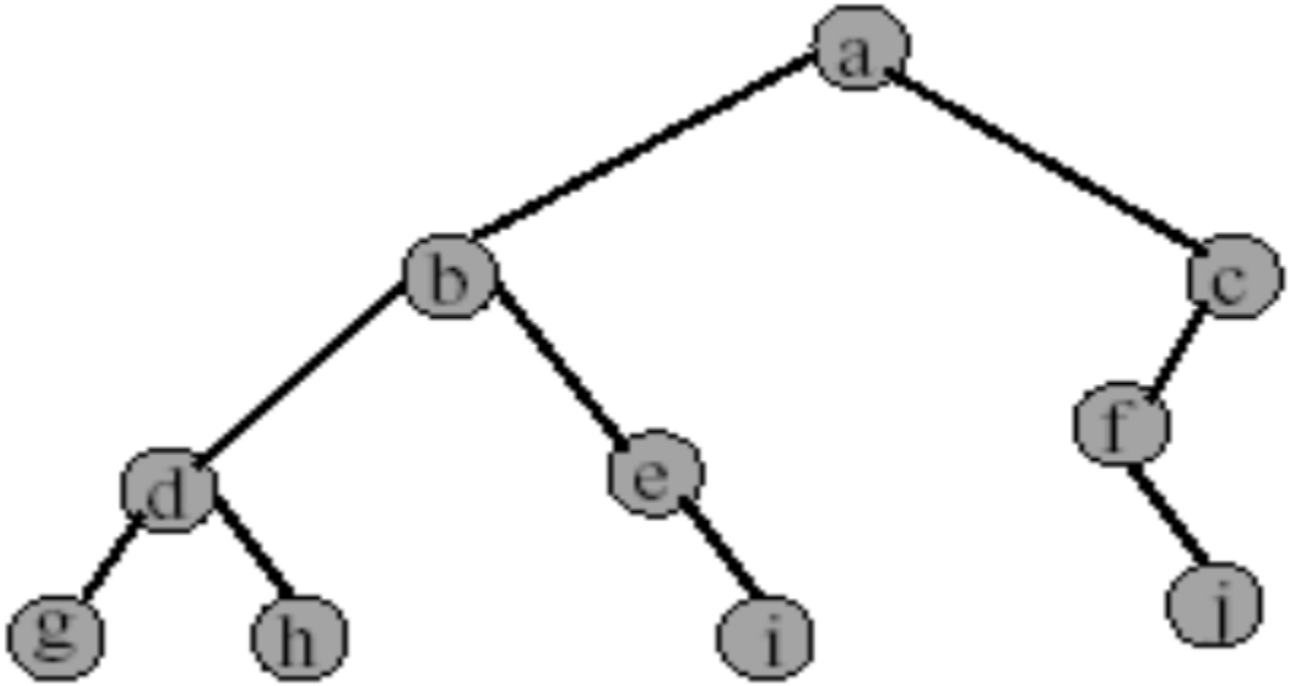
Postorder of Expression Tree



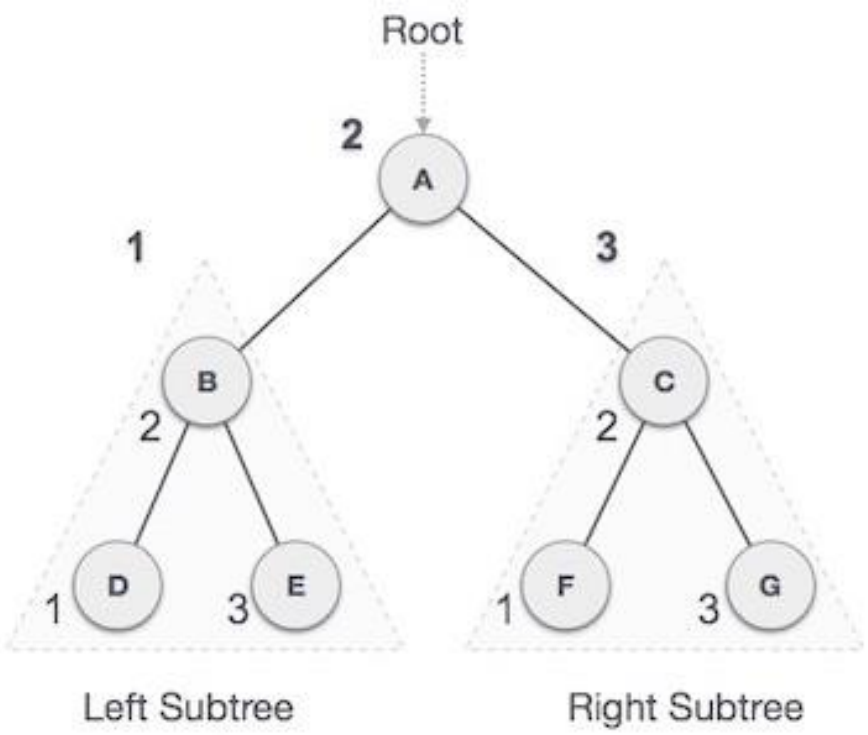
a b + c d - * e f + /

Gives postfix form of expression.

Level Order Example



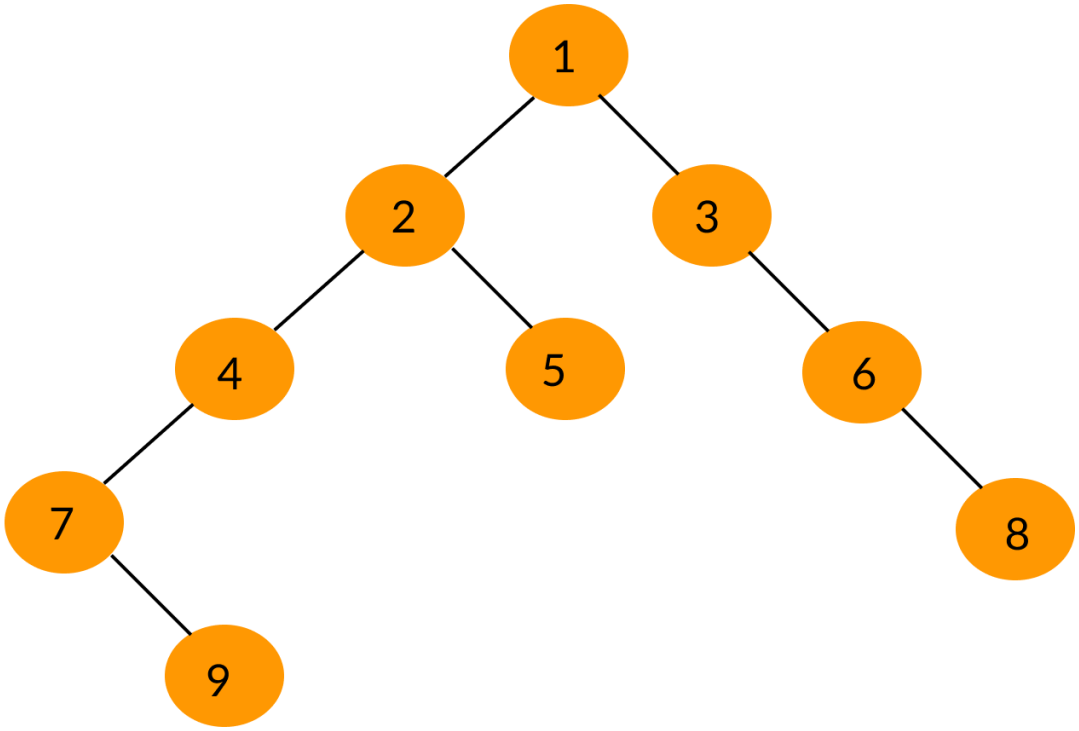
- Add and delete nodes from a queue
- Output: a b c d e f g h i j



In-Order : ***D → B → E → A → F → C → G***

Pre-Order : ***A → B → D → E → C → F → G***

Post-Order : ***D → E → B → F → G → C → A***



Inorder Traversal: 7 9 4 2 5 1 3 6 8

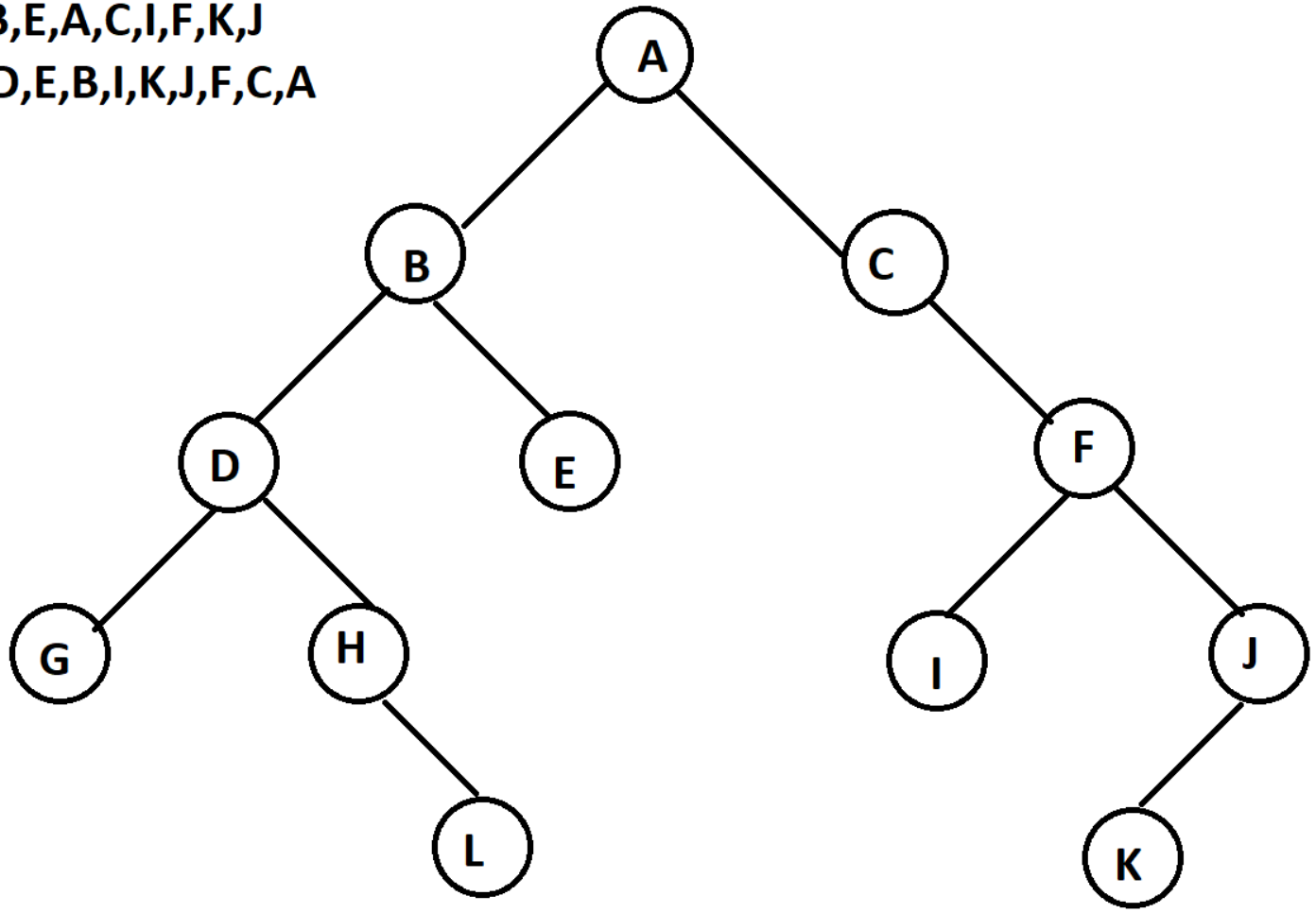
Preorder Traversal: 1 2 4 7 9 5 3 6 8

Postorder Traversal: 9 7 4 5 2 8 6 3 1

Pre-Order : A,B,D,G,H,L,E,C,F,I,J,K

In-Order : G,D,H,L,B,E,A,C,I,F,K,J

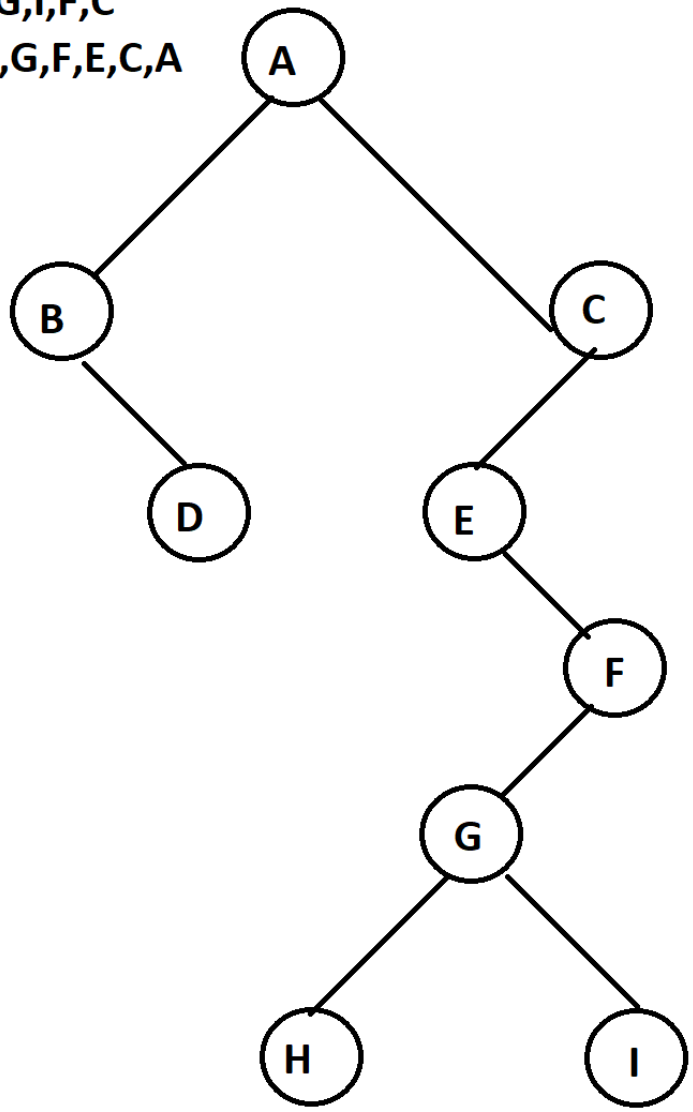
Post-Order : G,L,H,D,E,B,I,K,J,F,C,A



Pre-Order : A,B,D,C,E,F,G,H,I

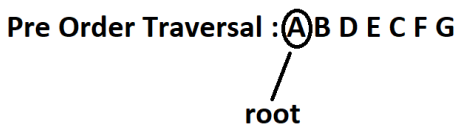
In-Order: B,D,A,E,H,G,I,F,C

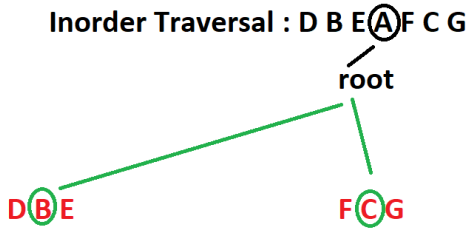
Post-OrdeR : D,B,H,I,G,F,E,C,A

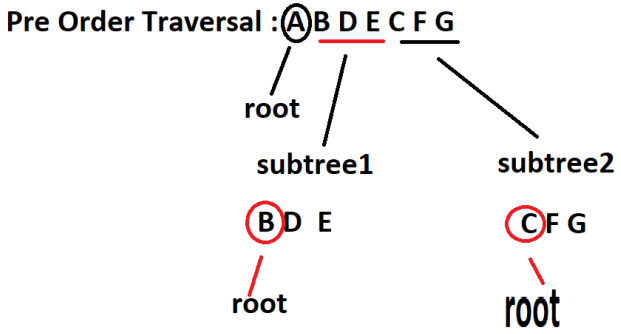


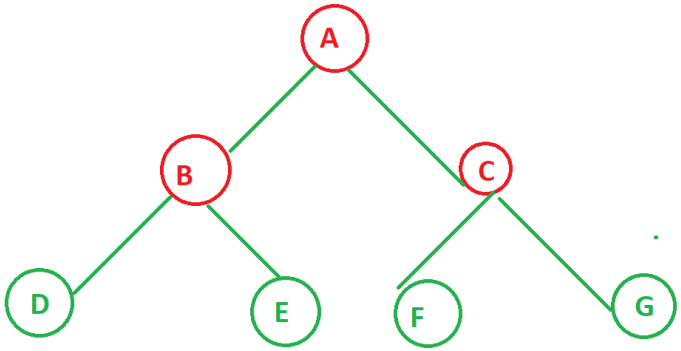
Converting a Binary Tree From Traversal Results

Inorder Traversal : D B E A F C G
Pre Order Traversal : A B D E C F G



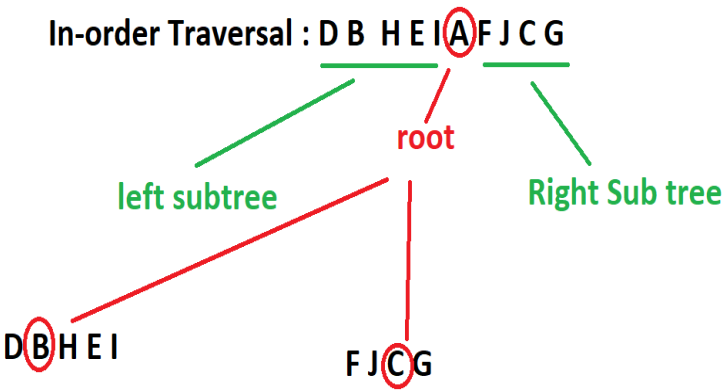




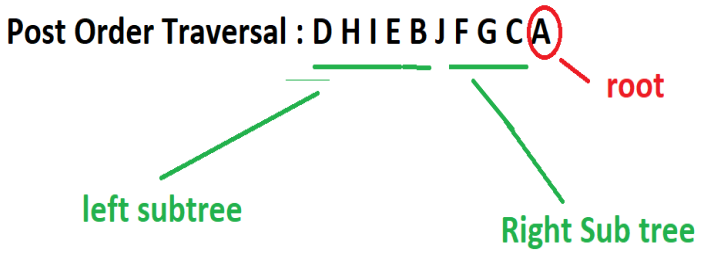


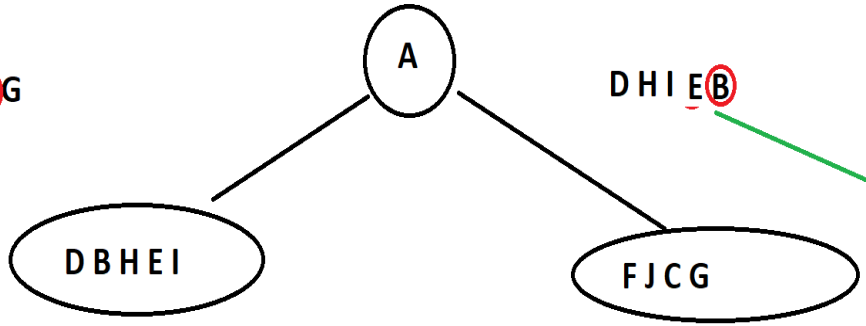
Constructing Binary tree from traversal Results

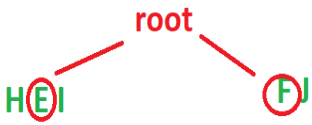
In-order Traversal : D B H E I A F J C G

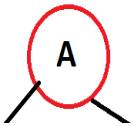


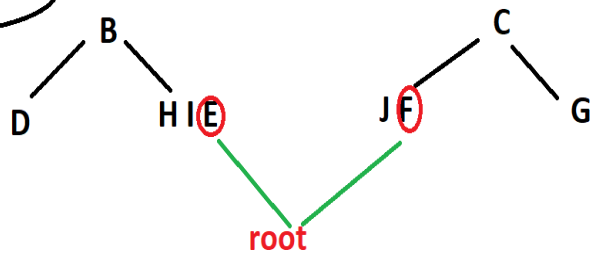
In Post Order traversal the last element id the root

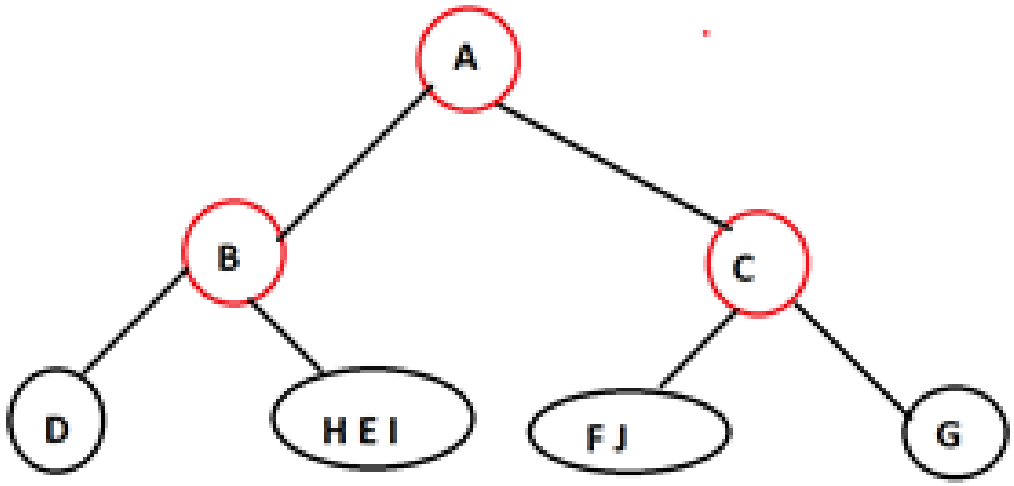


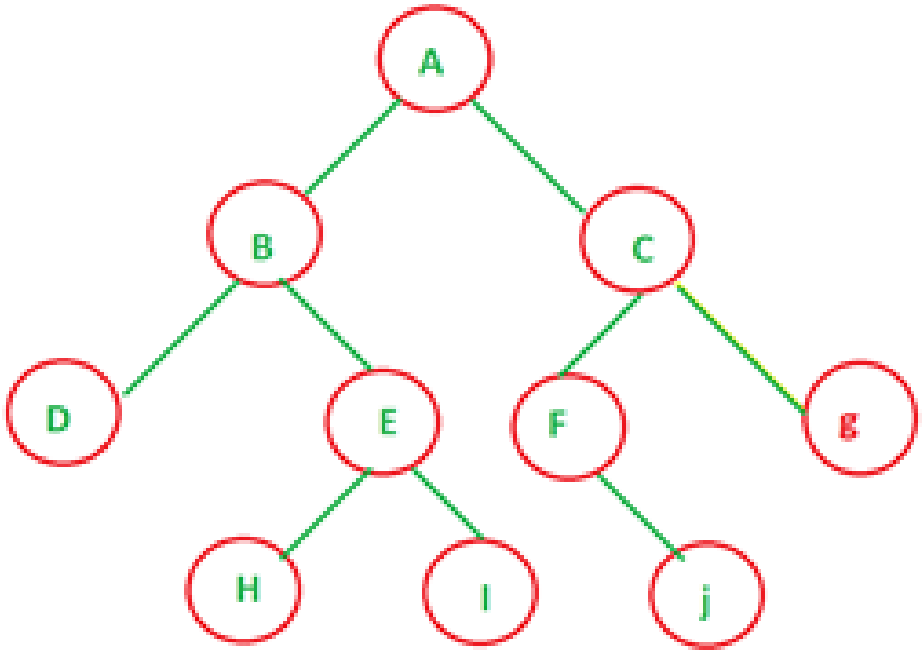








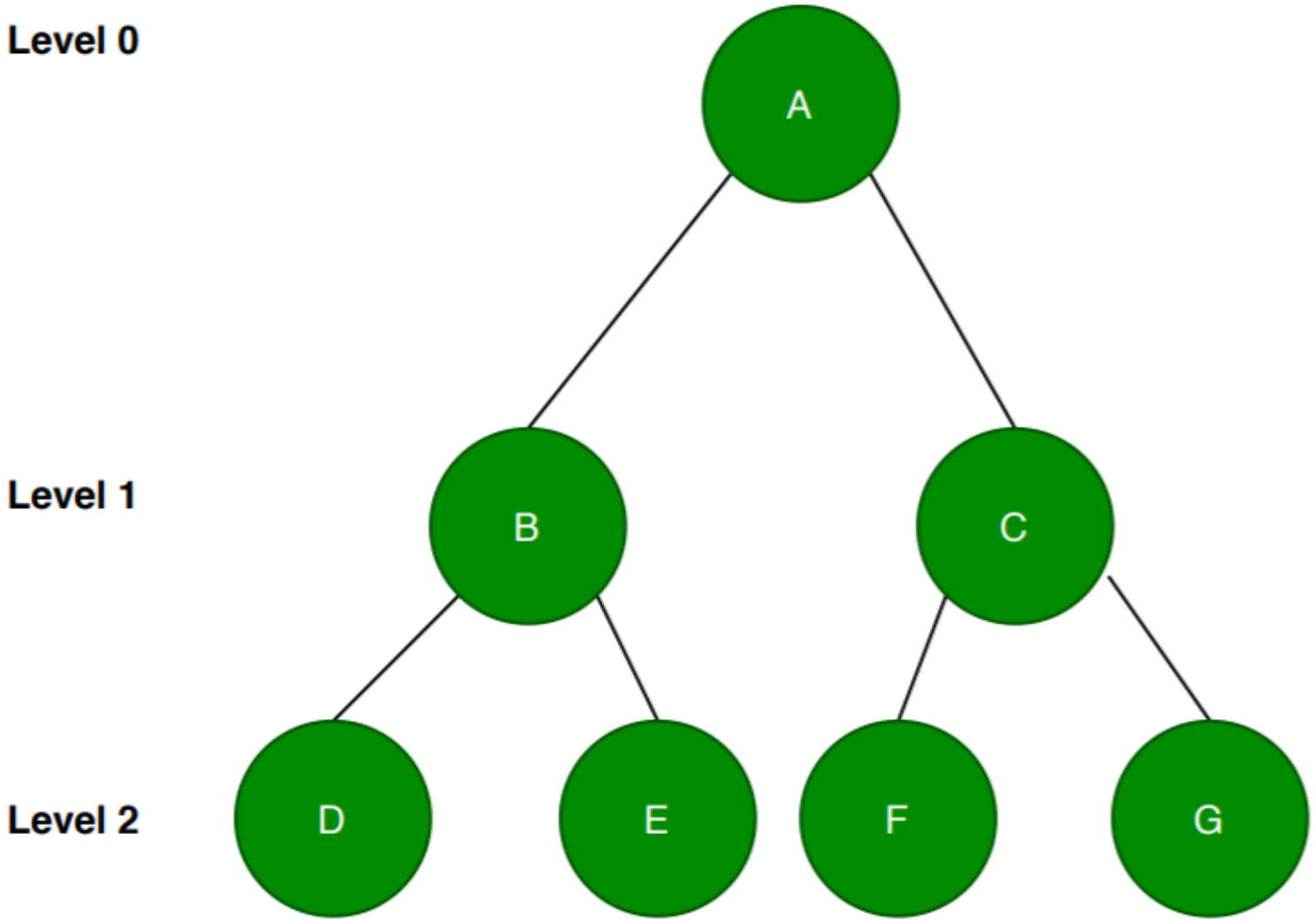

A
B
C
D
H E I
F J
G


A
B
C
D
E
F
G
H
I
j

Full Binary Tree

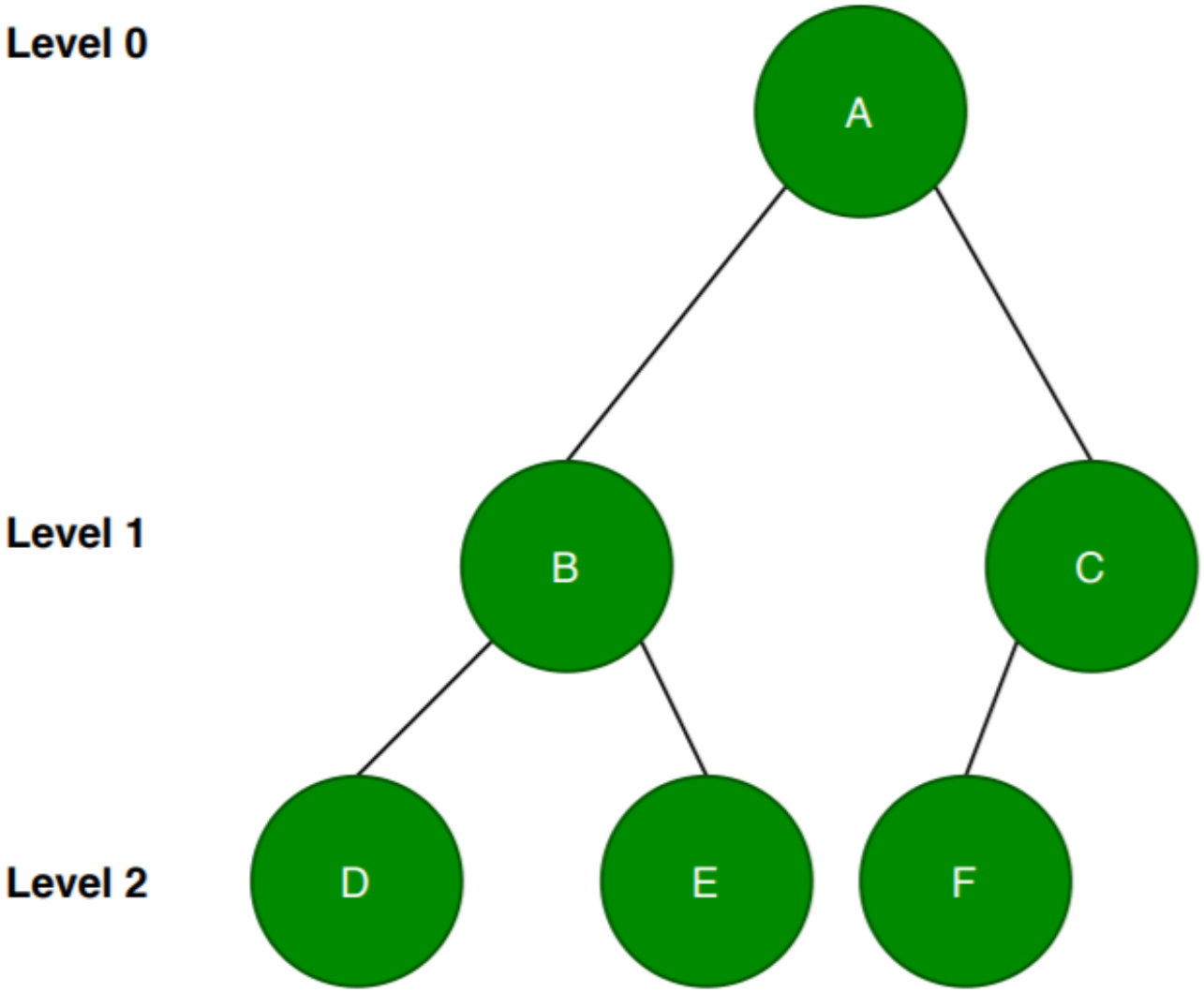
A full binary tree is also known **as 2-tree** in which **every node other than the leaf nodes has two child nodes**. It means **all the leaf nodes should be at the same level and all other internal nodes should contain two child nodes each.**

Example



If we look at this tree, we can see that it has two nodes for all the internal nodes except the leaf nodes. In addition to that, all the leaf nodes ***D, E, F, G*** are one the same level. Hence, we can say that this is an example of a full binary tree.

Example:



If we check this tree, the node ***C* is an internal node here and is not a leaf node. Although all the leaf nodes are on the same level, node *C* has only one child node which doesn't satisfy the definition.** Therefore, this example is not a full binary tree.

Complete Binary Tree

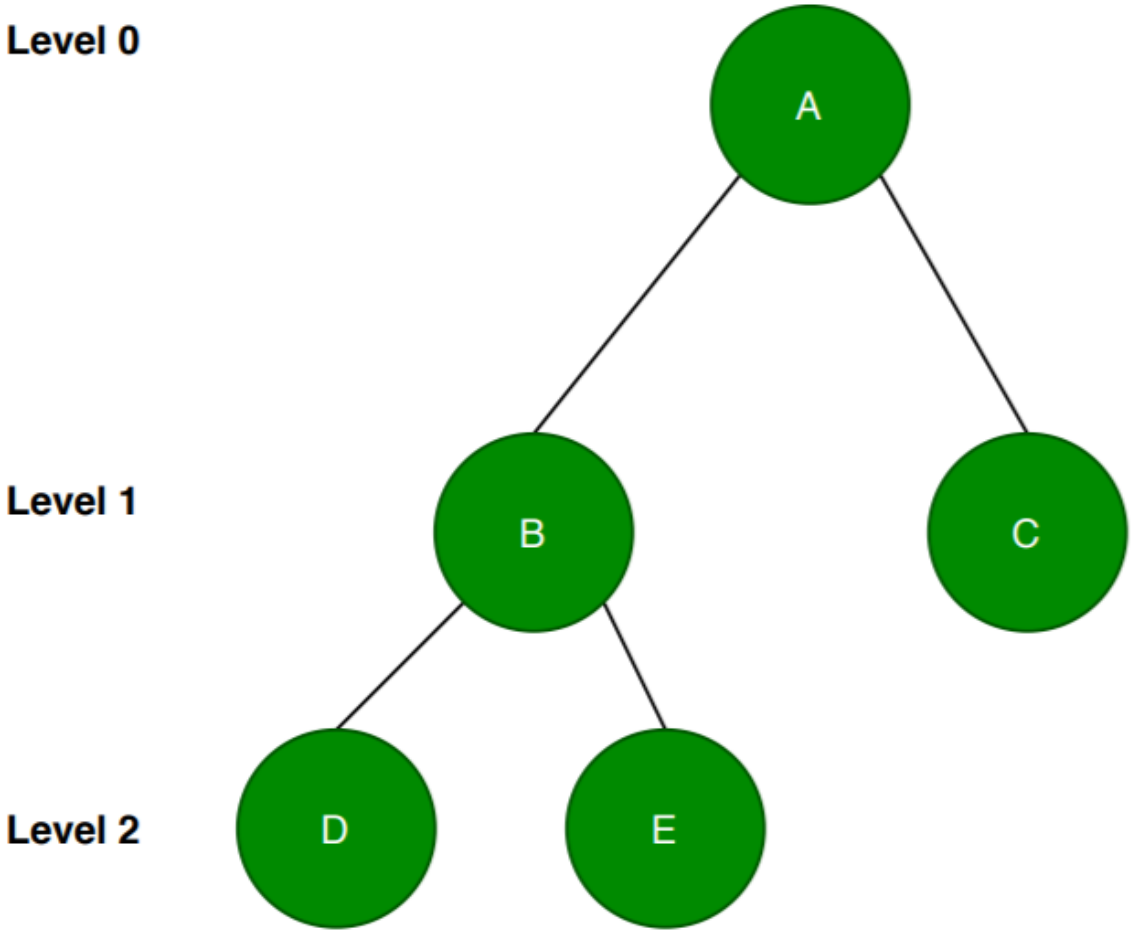
It follows the definition of a binary tree with some additional conditions. In a complete binary tree, **all the levels of a tree are filled entirely except the last level.** In the **last level, nodes might or might not be filled fully**. Also, let's note that all the nodes should be filled from the left.

A complete binary tree has two child nodes at each level. In the definition, we mentioned **all the levels are filled completely except at the last level.** Let's denote the level of a tree by ***L*. If a binary tree is completely filled then at each level we'll have 2^L nodes**.

In our case, **it is absolutely mandatory to contain 2^L nodes at each level except the last level.** Another point we mentioned in the definition is that **we fill the nodes from the left**. So **it is not allowed for a node to have a right child node without having a left child node.**

Example

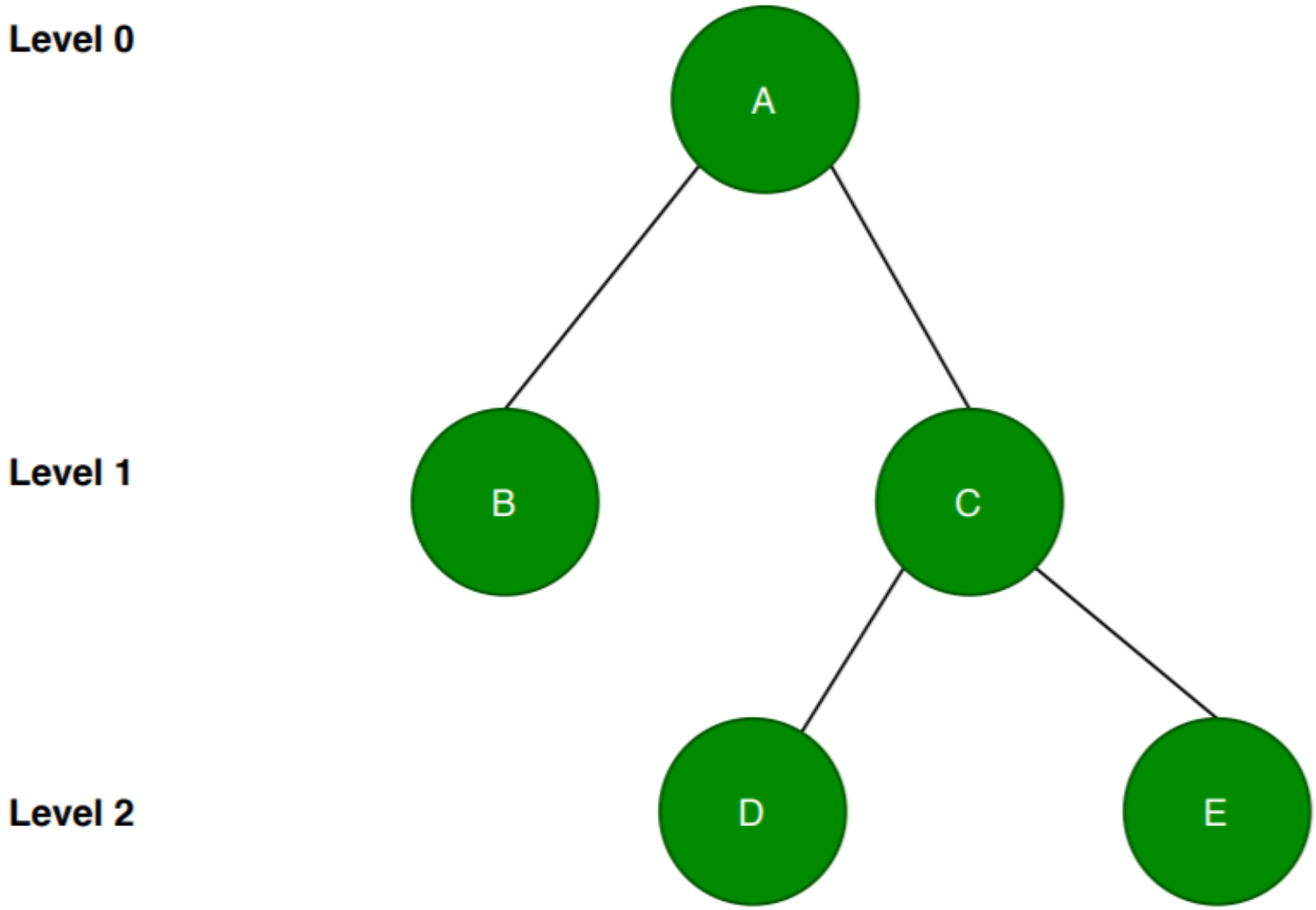
Now we know the theory, let's verify the definition with the help of a couple of examples:



According to the definition, all the levels except the last level should be completely filled. Here, the level ***2*** is the last level. The tree has the root node ***A*** at level ***0*** and two nodes ***B*** and ***C*** at level ***1***. Till now, it satisfies the definition. Also, let's note that the nodes are filled from the left.

At level ***2***, there are ***2*** nodes. But most importantly, the nodes are filled from left. Therefore we can conclude that it is a complete binary tree.

Example:



First, we'll check if the given tree has the required number of nodes at each level except the last level which is level ***2*** in this case. We can see that the given tree satisfies this condition.

Next, we'll check whether the nodes are filled from the left or not. At level ***2***, the node ***B*** has no child nodes. That's fine according to the definition.

But the problem with this tree is that the node ***C*** has two child nodes. Now according to the definition, the nodes need to be filled from the left. So there should be a left child node for the node ***B*** which is missing here. Therefore, this example doesn't follow the definition. Hence, it is not a complete binary tree.

Application

- We can use it in the heap data structure. In computer science, heaps can be of two types: max-heap and min-heap. Algorithms like Heap Sort uses it for sorting.
- It is also used in implementing priority queues and external sorting algorithms.

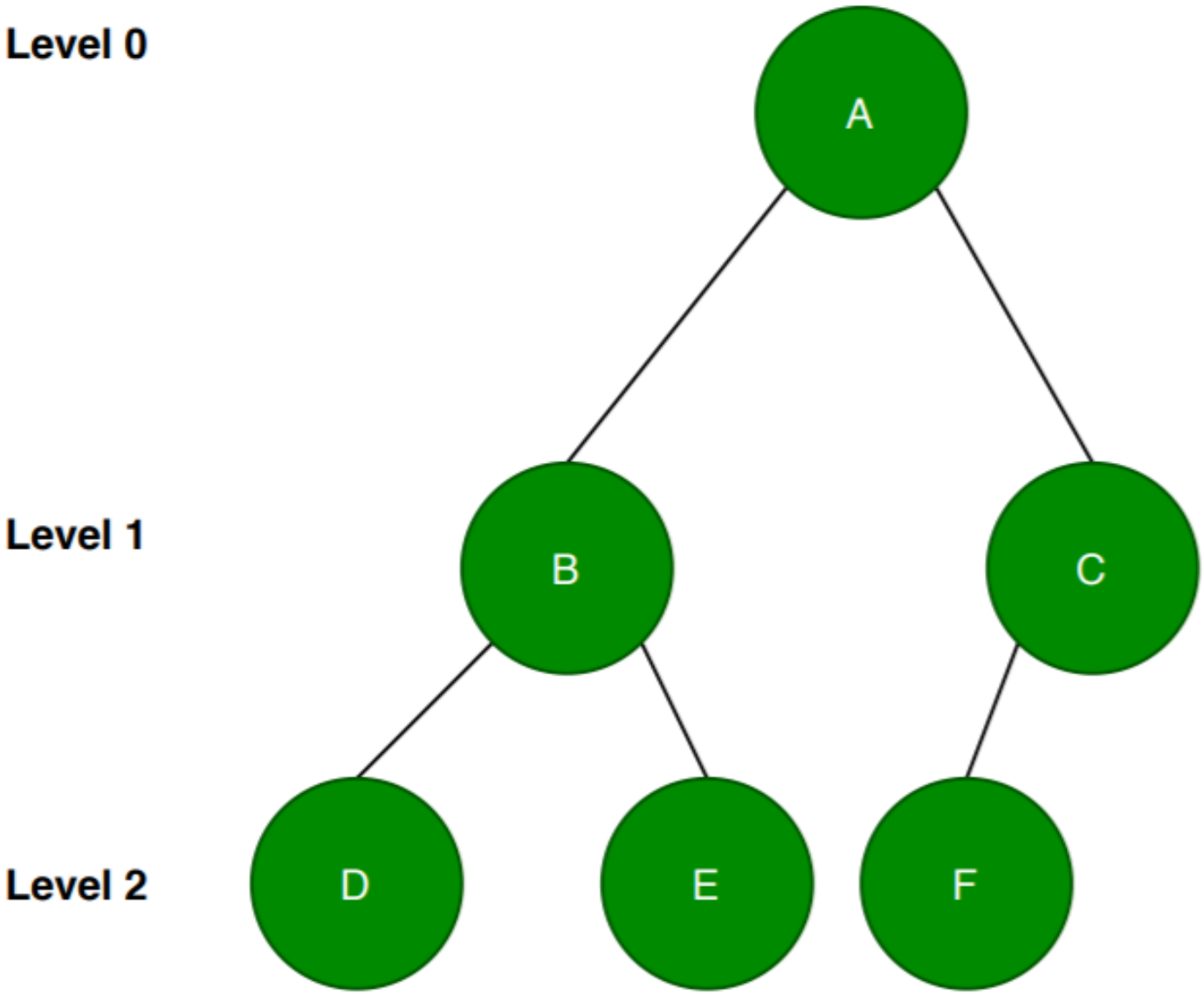
Almost Complete Binary Tree

An almost complete binary tree is a **special kind of binary tree where insertion takes place level by level and from left to right order at each level and the last level is not filled fully always. It also contains 2^L nodes at each level except the last level.**

The main point here is that **if we want to insert any node at the lowest level of the tree, it should be inserted from the left**. All the **internal nodes should be completely filled.**

An **almost complete tree is also a complete binary tree**.

Example

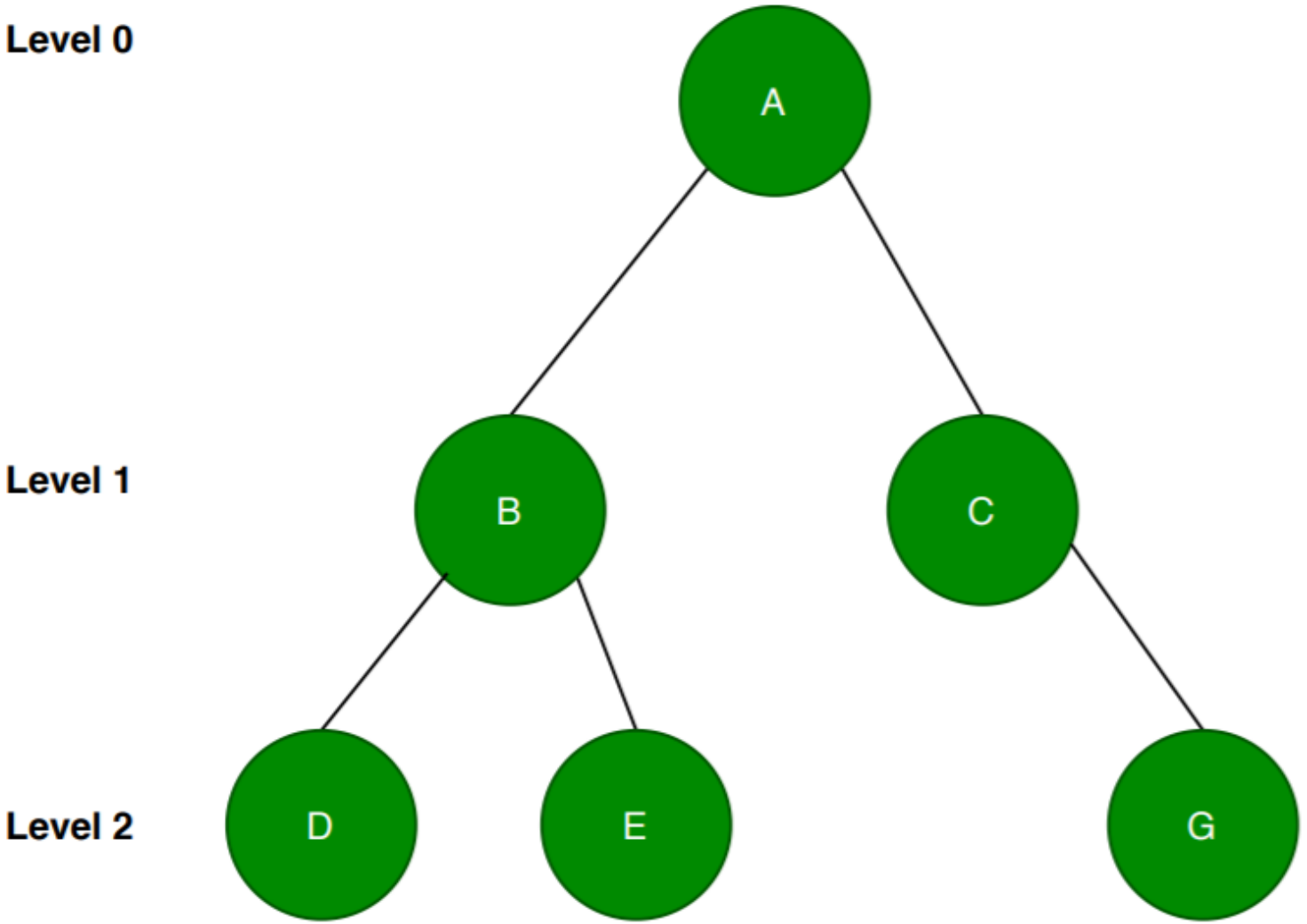


Notice that this example is the same as the first example of a full binary tree with the missing node ***G***. So here, we deleted the node ***G*** at the level ***2***. At this point, if we observe, it satisfies the definition of a complete binary tree. The question is whether it is an almost complete binary tree?

The answer would be yes. Let's discuss further. This example satisfies the definition as it has the required number of nodes at level ***0*** and ***1*** . Level is the last level here. Therefore, it is not required to have all the 2^L nodes at level ***2*** .

Now, what is interesting to analyze is whether the nodes at the lowest level are present from left to the right direction or not. So here the node ***B*** has two child nodes ***D, E*** and the node ***C*** has one child node ***F*** but most importantly it is left child node. Therefore it satisfies the definition of an almost complete binary tree.

Example:



Again, notice that this is exactly the same as the first example but here we've deleted the node ***F*** from the tree. According to the definition, we should insert the nodes from left to right at each level. Here, in this case, we can see that the node ***C*** doesn't have any left child node. We deleted the left child although the node has a right child node. This is not allowed according to the definition.

Therefore, this is not an almost binary tree nor complete.

A High-Level Difference

Parameter	Complete Binary Tree	Almost Complete Binary Tree
Definition	When the tree is complete then the last level might or might not be full.	In an almost complete binary tree the last level is not full for sure.
Property	A complete binary tree may or may not be an almost complete binary tree.	An almost complete binary tree will always be a complete binary tree.
Application	Heap data structure.	It doesn't have any applications. If needed, a complete binary tree is used