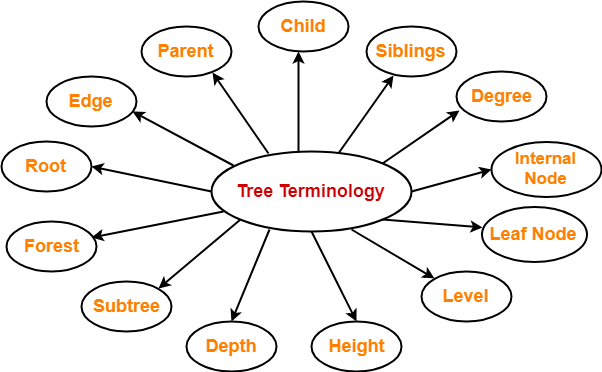
DS Unit 4

Tree

A tree consists of a root, and zero or more subtrees T1, T2, … , Tk such that there is an edge from the root of the tree to the root of each subtree.

**Tree Terminology-**

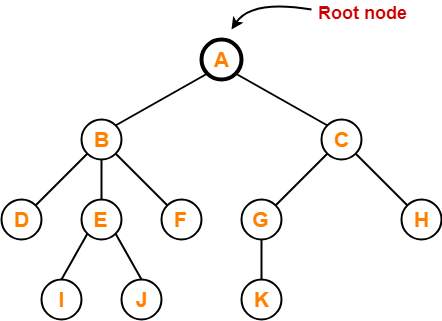
The important terms related to tree data structure are-



**1. Root-**

* The first node from where the tree originates is called as a **root node**.
* In any tree, there must be only one root node.
* We can never have multiple root nodes in a tree data structure.

**Example-**

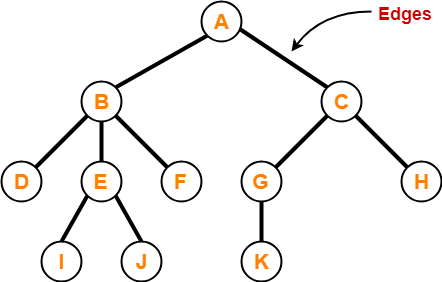


Here, node A is the only root node.

**2. Edge-**

* The connecting link between any two nodes is called as an **edge**.
* In a tree with n number of nodes, there are exactly (n-1) number of edges.

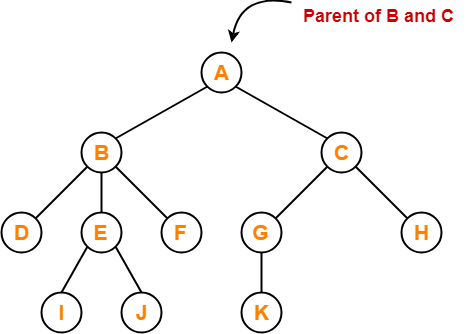
**Example-**



**3. Parent-**

* The node which has a branch from it to any other node is called as a **parent node**.
* In other words, the node which has one or more children is called as a parent node.
* In a tree, a parent node can have any number of child nodes.

**Example-**



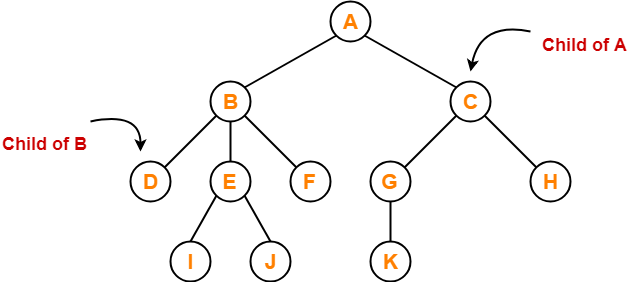
Here,

* Node A is the parent of nodes B and C
* Node B is the parent of nodes D, E and F
* Node C is the parent of nodes G and H
* Node E is the parent of nodes I and J
* Node G is the parent of node K

**4. Child-**

* The node which is a descendant of some node is called as a **child node**.
* All the nodes except root node are child nodes.

**Example-**



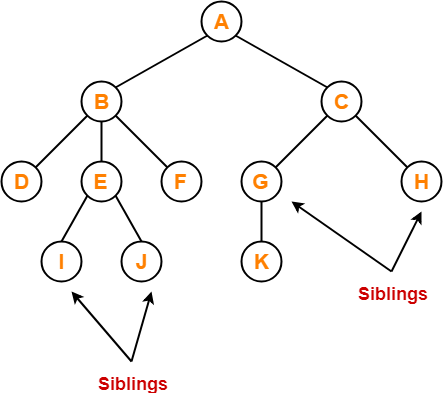
Here,

* Nodes B and C are the children of node A
* Nodes D, E and F are the children of node B
* Nodes G and H are the children of node C
* Nodes I and J are the children of node E
* Node K is the child of node G

**5. Siblings-**

* Nodes which belong to the same parent are called as **siblings**.
* In other words, nodes with the same parent are sibling nodes.

**Example-**



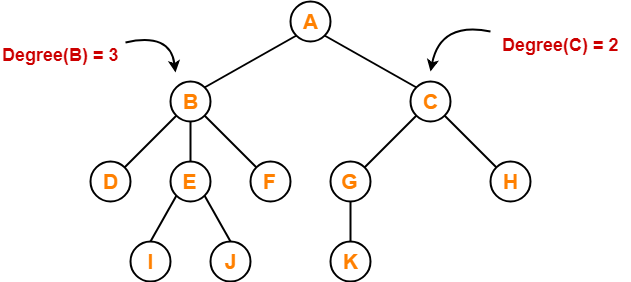
Here,

* Nodes B and C are siblings
* Nodes D, E and F are siblings
* Nodes G and H are siblings
* Nodes I and J are siblings

**6. Degree-**

* **Degree of a node** is the total number of children of that node.
* **Degree of a tree** is the highest degree of a node among all the nodes in the tree.

**Example-**



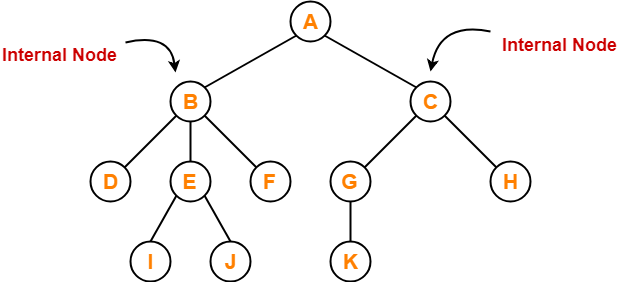
Here,

* Degree of node A = 2
* Degree of node B = 3
* Degree of node C = 2
* Degree of node D = 0
* Degree of node E = 2
* Degree of node F = 0
* Degree of node G = 1
* Degree of node H = 0
* Degree of node I = 0
* Degree of node J = 0
* Degree of node K = 0

**7. Internal Node-**

* The node which has at least one child is called as an **internal node**.
* Internal nodes are also called as **non-terminal nodes**.
* Every non-leaf node is an internal node.

**Example-**

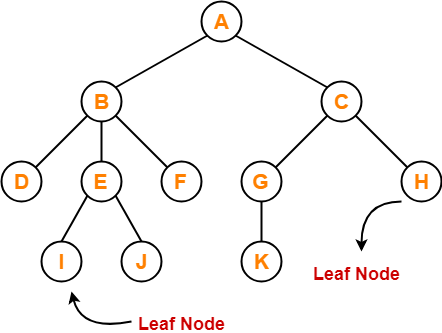


Here, nodes A, B, C, E and G are internal nodes.

**8. Leaf Node-**

* The node which does not have any child is called as a **leaf node**.
* Leaf nodes are also called as **external nodes** or **terminal nodes**.

**Example-**

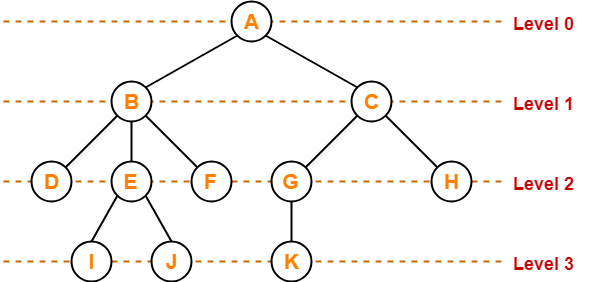


Here, nodes D, I, J, F, K and H are leaf nodes.

**9. Level-**

* In a tree, each step from top to bottom is called as **level of a tree**.
* The level count starts with 0 and increments by 1 at each level or step.

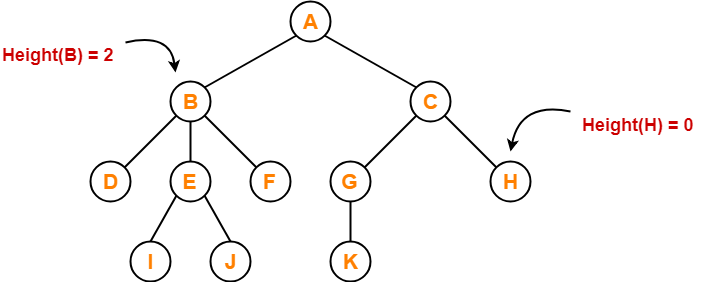
**Example-**



**10. Height-**

* Total number of edges that lies on the longest path from any leaf node to a particular node is called as **height of that node**.
* **Height of a tree** is the height of root node.
* Height of all leaf nodes = 0

**Example-**



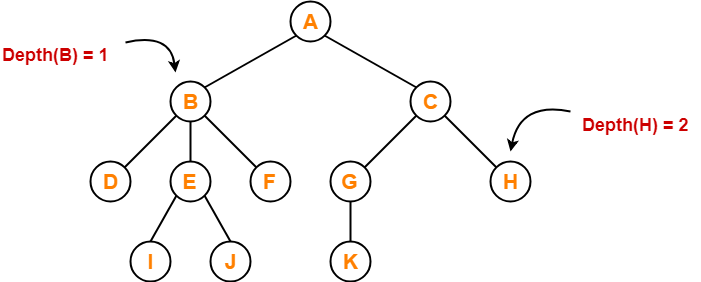
Here,

* Height of node A = 3
* Height of node B = 2
* Height of node C = 2
* Height of node D = 0
* Height of node E = 1
* Height of node F = 0
* Height of node G = 1
* Height of node H = 0
* Height of node I = 0
* Height of node J = 0
* Height of node K = 0

**11. Depth-**

* Total number of edges from root node to a particular node is called as **depth of that node**.
* **Depth of a tree** is the total number of edges from root node to a leaf node in the longest path.
* Depth of the root node = 0
* The terms “level” and “depth” are used interchangeably.

**Example-**



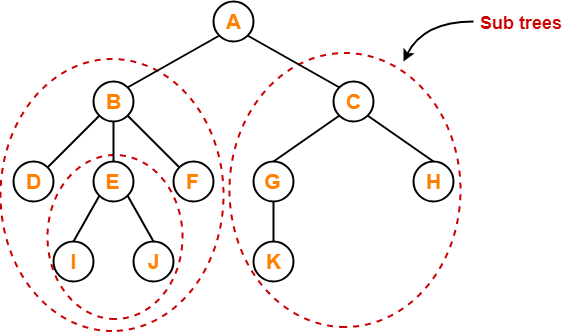
Here,

* Depth of node A = 0
* Depth of node B = 1
* Depth of node C = 1
* Depth of node D = 2
* Depth of node E = 2
* Depth of node F = 2
* Depth of node G = 2
* Depth of node H = 2
* Depth of node I = 3
* Depth of node J = 3
* Depth of node K = 3

**12. Subtree-**

* In a tree, each child from a node forms a **subtree** recursively.
* Every child node forms a subtree on its parent node.

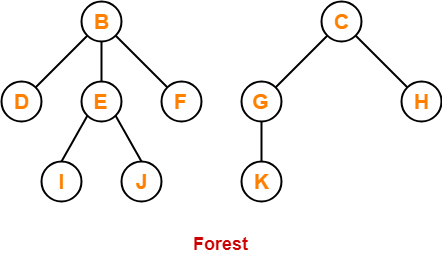
**Example-**



**13. Forest-**

A forest is a set of disjoint trees.

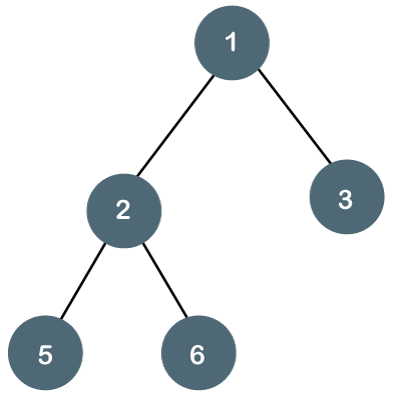
**Example-**



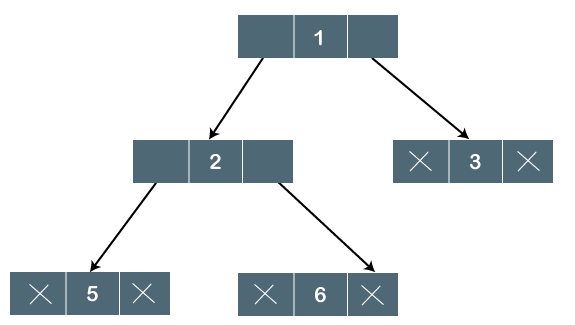
# **Binary Tree**

The Binary tree means that the node can have maximum two children. Here, binary name itself suggests that 'two'; therefore, each node can have either 0, 1 or 2 children.

**Let's understand the binary tree through an example.**



The above tree is a binary tree because each node contains the utmost two children. The logical representation of the above tree is given below:



In the above tree, node 1 contains two pointers, i.e., left and a right pointer pointing to the left and right node respectively. The node 2 contains both the nodes (left and right node); therefore, it has two pointers (left and right). The nodes 3, 5 and 6 are the leaf nodes, so all these nodes contain **NULL** pointer on both left and right parts.

35

**Properties of Binary Tree**

* At each level of i, the maximum number of nodes is 2i.
* The height of the tree is defined as the longest path from the root node to the leaf node. The tree which is shown above has a height equal to 3. Therefore, the maximum number of nodes at height 3 is equal to (1+2+4+8) = 15. In general, the maximum number of nodes possible at height h is (20 + 21 + 22+….2h) = 2h+1 -1.
* The minimum number of nodes possible at height h is equal to **h+1**.
* If the number of nodes is minimum, then the height of the tree would be maximum. Conversely, if the number of nodes is maximum, then the height of the tree would be minimum.

# **Representation of a Binary Tree**

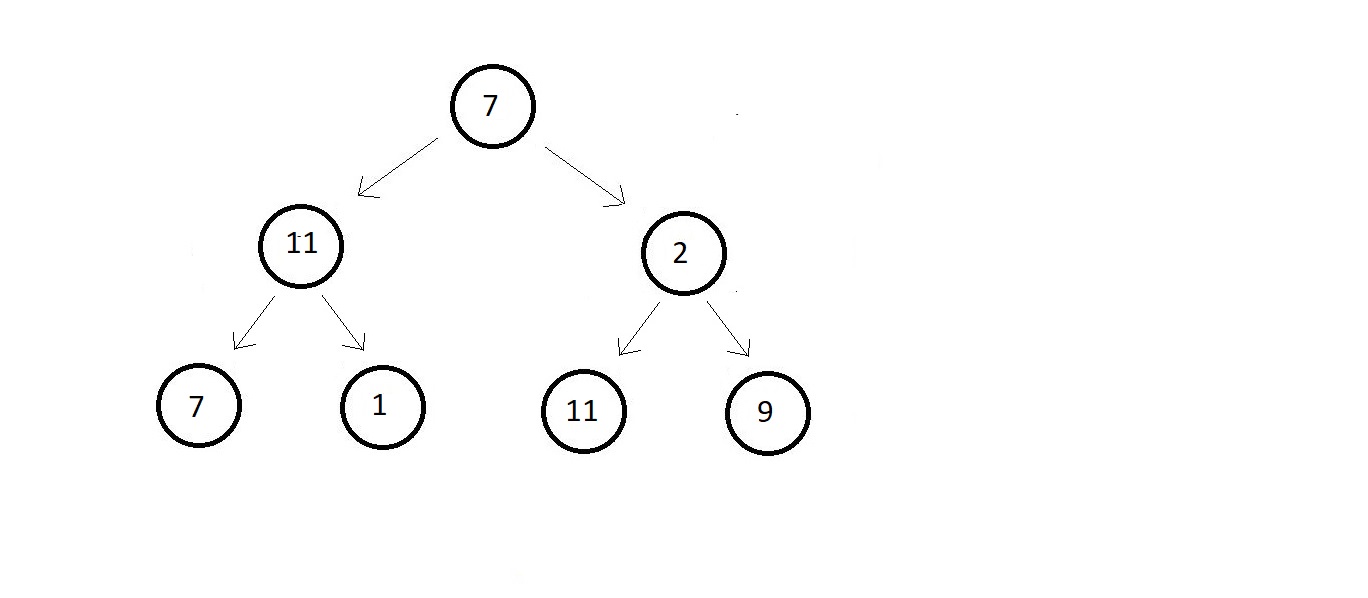
In today's lesson, we'll consider different techniques for representing binary trees in programming, and see which one best suits our needs.

So, the first way to represent a binary tree is by using arrays. We call this ‘array representation’. And this method is not very recommended for representing binary trees. You will very soon know why.

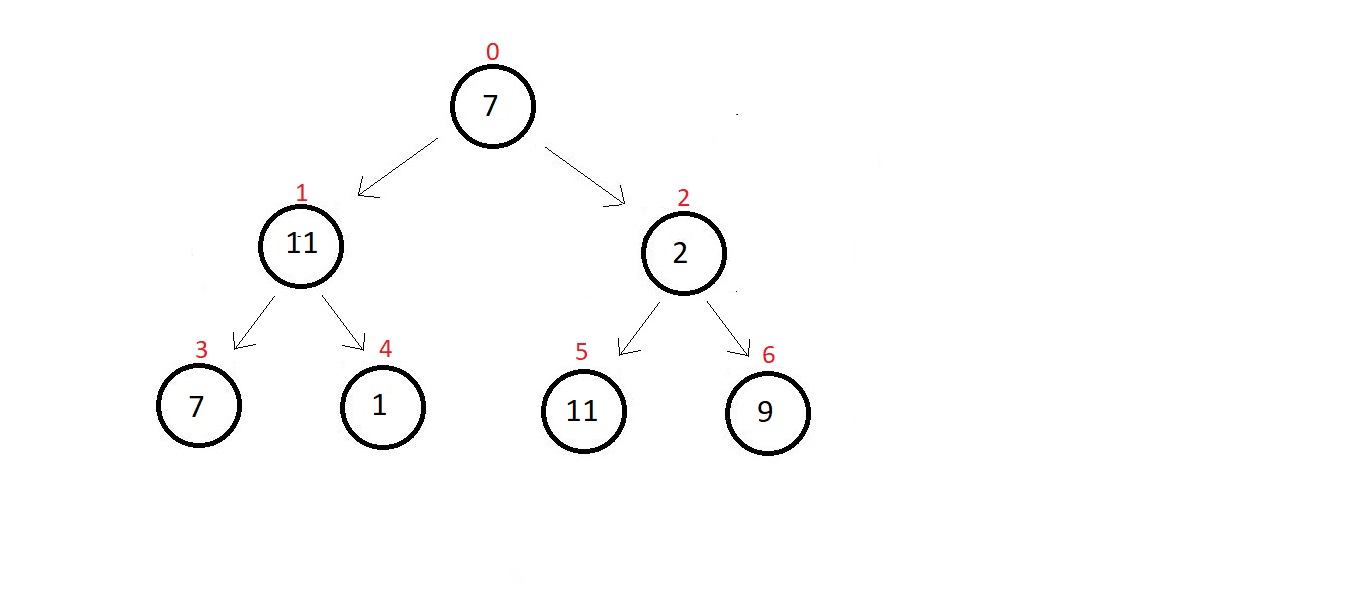
#### **Array representation of Binary trees:**

Arrays are linear data structures and for arrays to function, their size must be specified before elements are inserted into them. And this counts as the biggest demerit of representing binary trees using arrays. Suppose you declare an array of size 100, and after storing 100 nodes in it, you cannot even insert a single element further, regardless of all the spaces left in the memory.  Another way you’d say is to copy the whole thing again to a new array of bigger size but that is not considered a good practice.

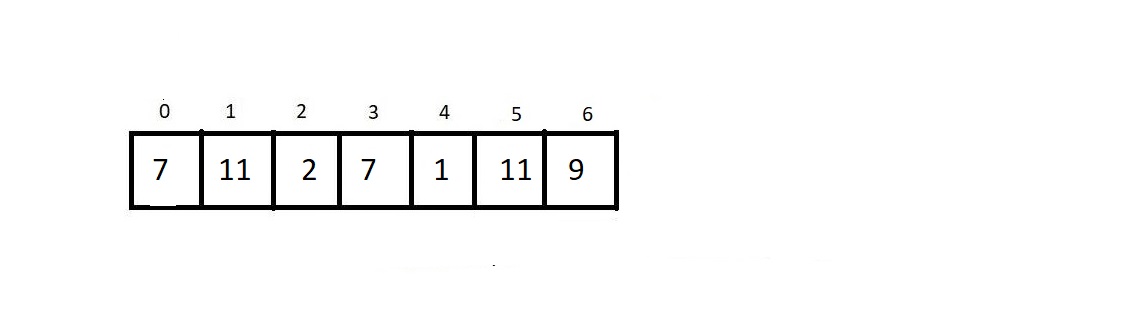
Anyways, we will use an array to represent a binary tree. Suppose we have a binary tree with 7 nodes.



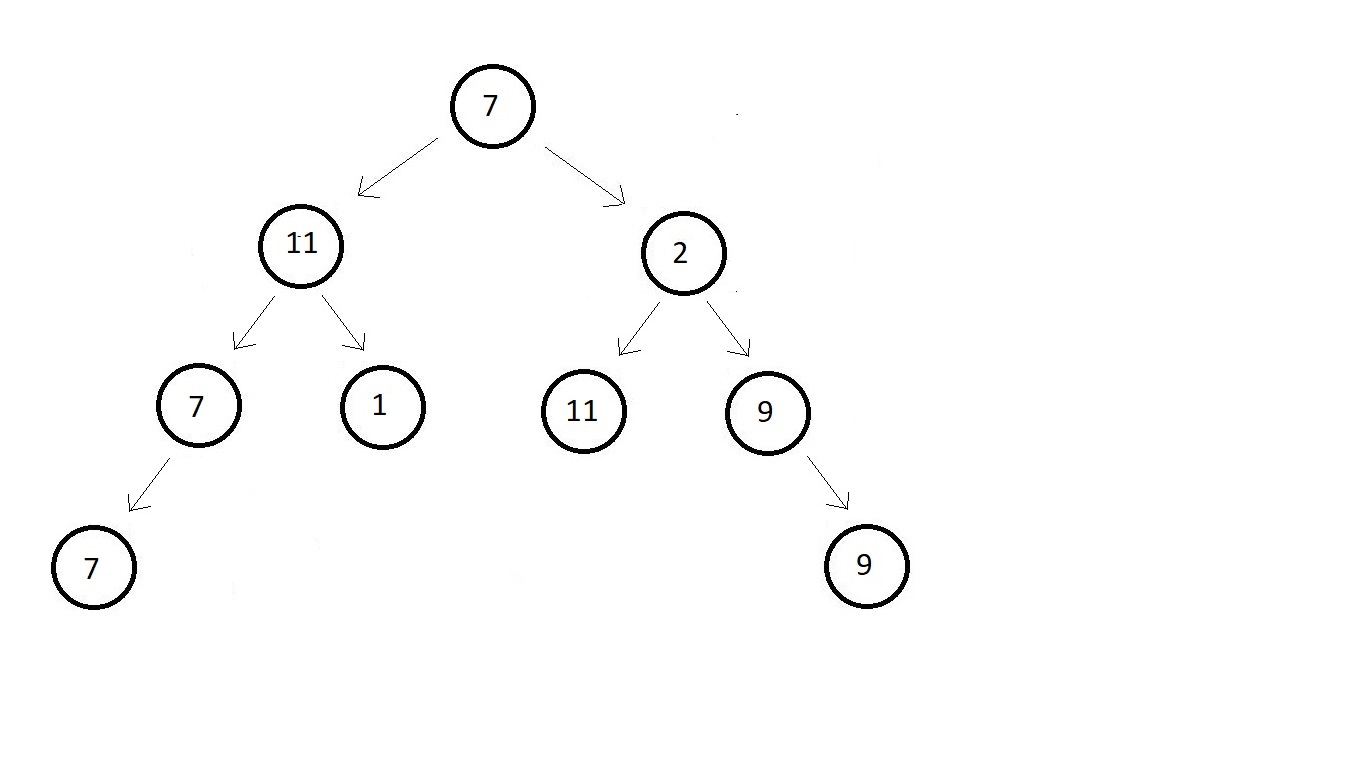
And there are actually a number of ways to represent these nodes via an array. I’ll use the most convenient one where we traverse each level starting from the root node and from left to right and mark them with the indices these nodes would belong to.



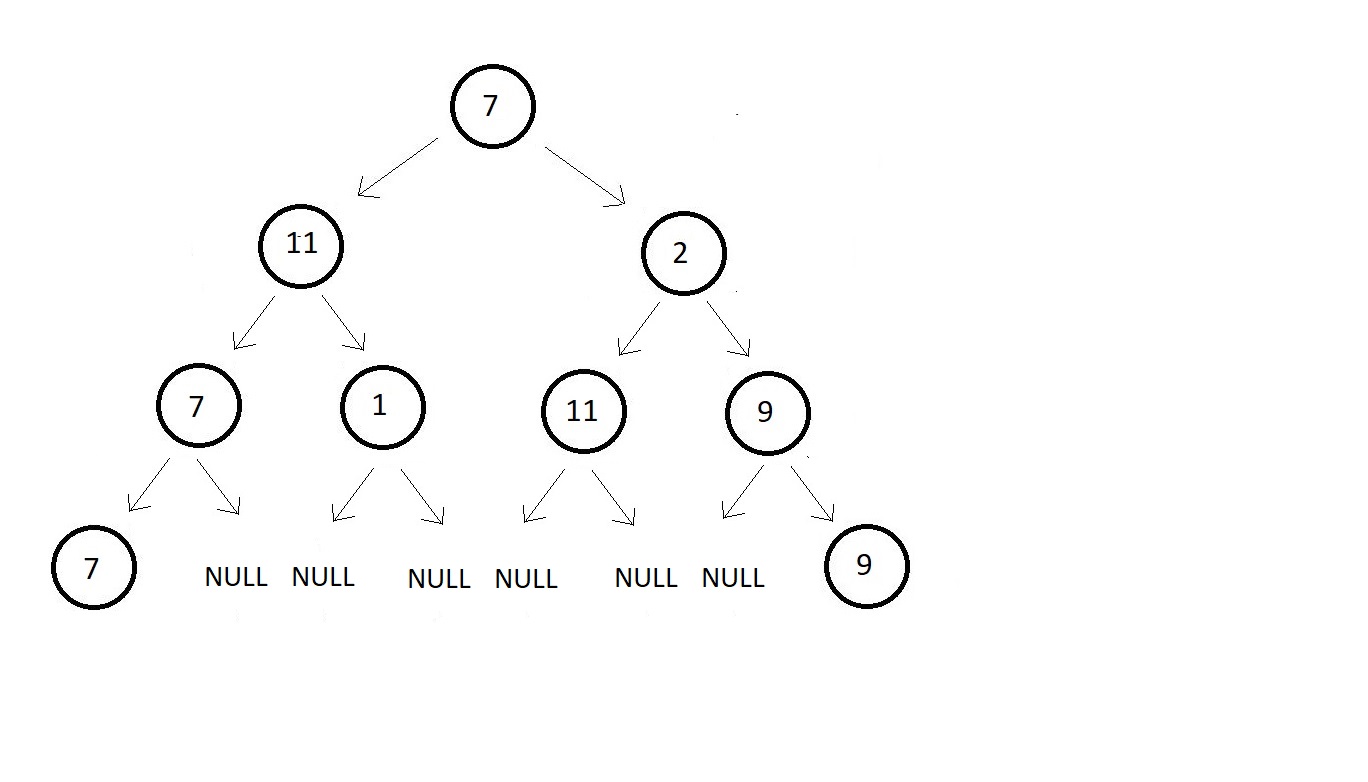
And now we can simply make an array of length 7 and store these elements at their corresponding indices.



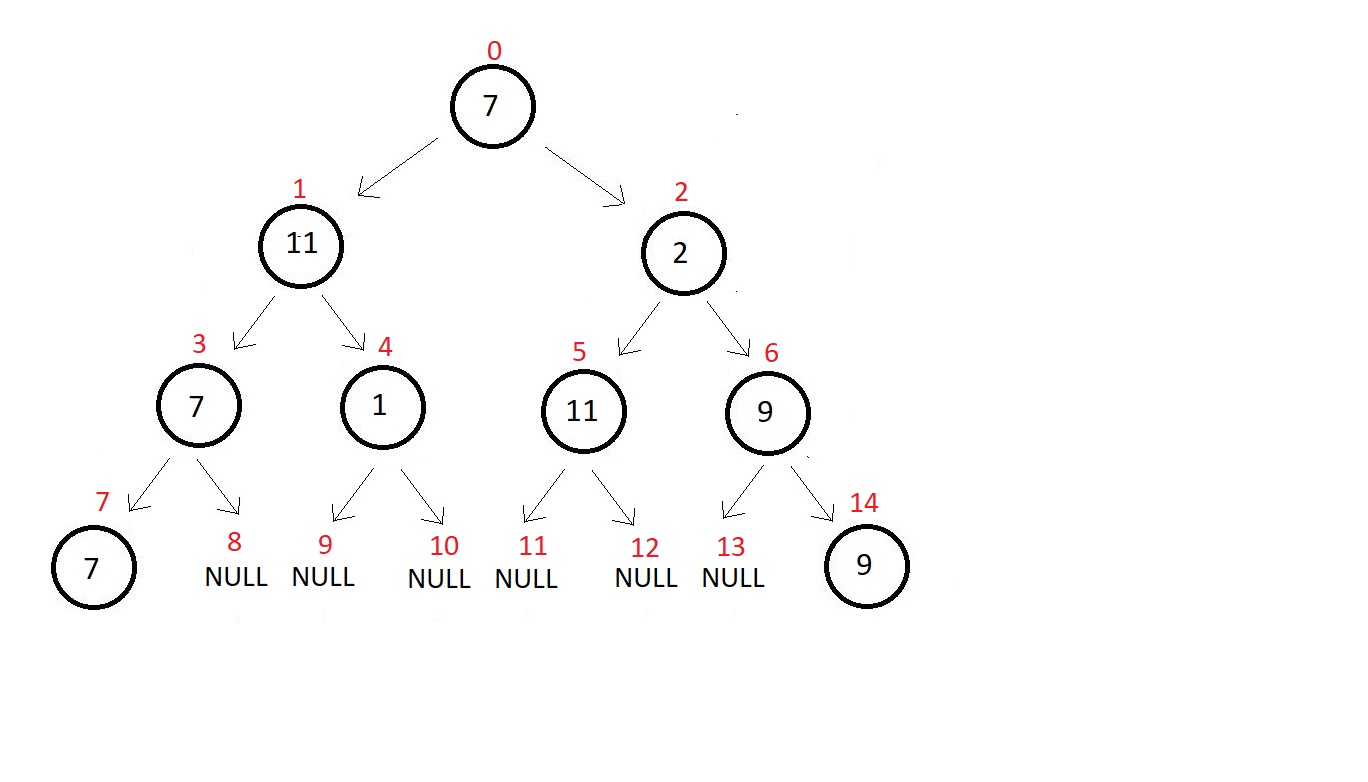
And you might be wondering about the cases where the binary is just not perfect. What if the last level has distributed leaves? Then let me tell you, there is a way out for that as well. Let’s consider one case here. A binary tree with 9 nodes, and the last two nodes on the extremities of the last level.



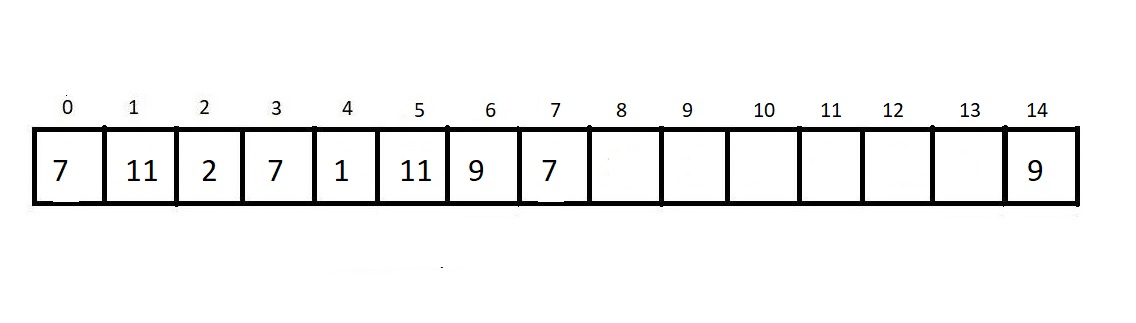
Here, while traversing we get stuck at the 8th index. We don’t know if declaring the last node as the 8th index element makes it a general representation of the tree or not. So, we simply make the tree perfect ourselves. We first assume the remaining vacant places to be NULL.



And now we can easily mark their indices from 0 to 14.



And the array representation of the tree looks something like this. It is an array of length 15.



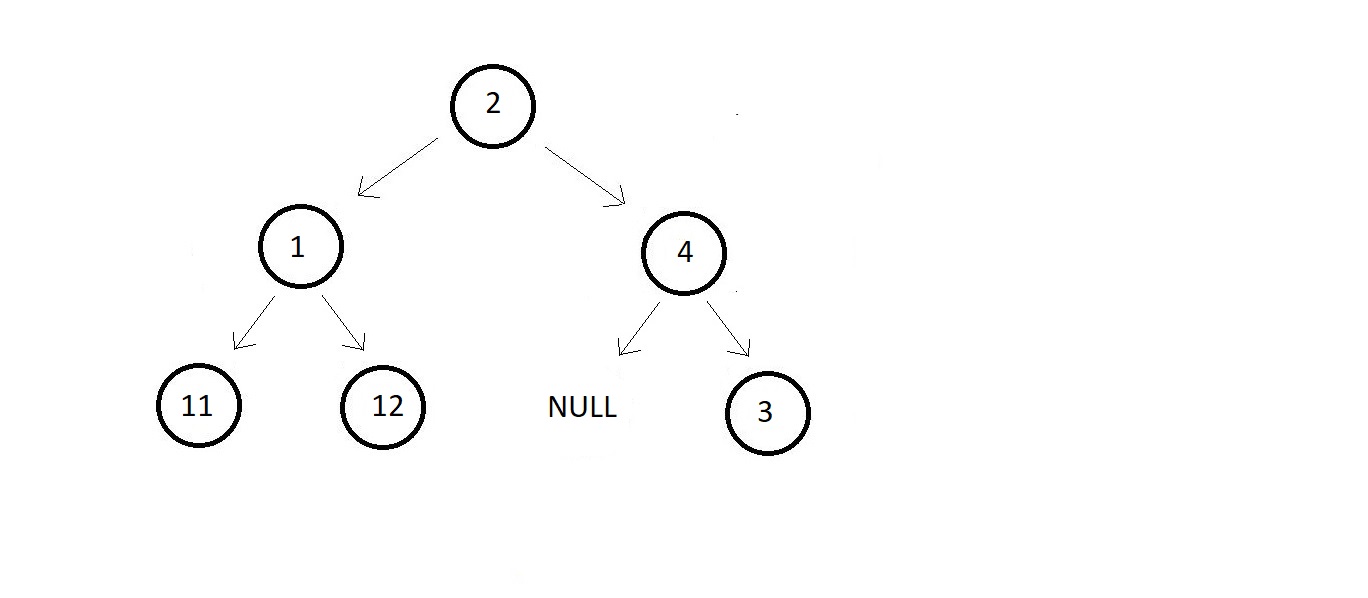
But was this even an efficient approach? Like Binary Trees are made only for efficient traversal and insertion and deletion and using an array for that really makes the process troublesome. Each of these operations becomes quite costly to accomplish. And that size constraint was already for making things worse. So overall, we would say that the array representation of a binary is not a very good choice. And what are the other options?

We have another method to represent binary trees called the linked representation of binary trees. Don’t confuse this with linked lists you have studied. And the reason why I am saying that is because linked lists are lists that are linear data structures.

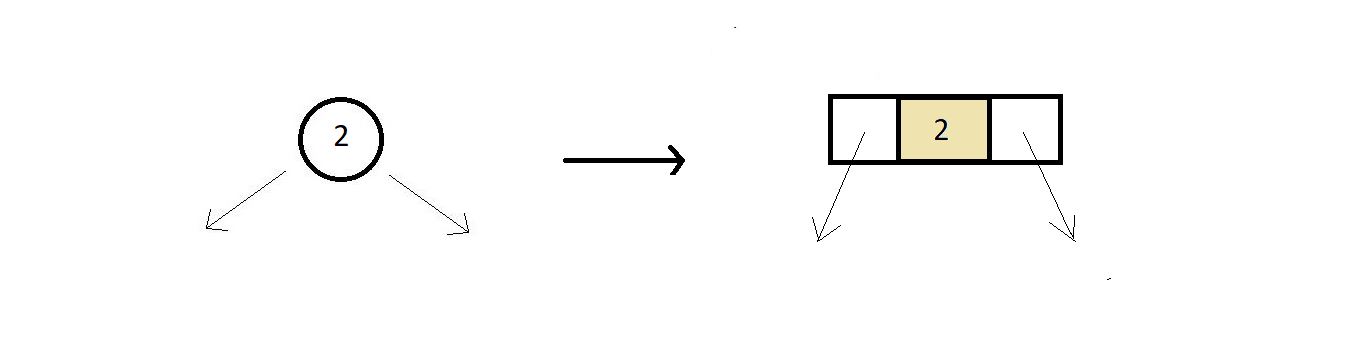
**Linked Representation of Binary Trees:**

This method of representing binary trees using linked nodes is considered the most efficient method of representation. For this, we use doubly-linked lists. I just hope you recall what doubly-linked lists are. We studied that here in the same playlist [Doubly Linked Lists Explained With Code in C Language](https://www.codewithharry.com/videos/data-structures-and-algorithms-in-hindi-21).

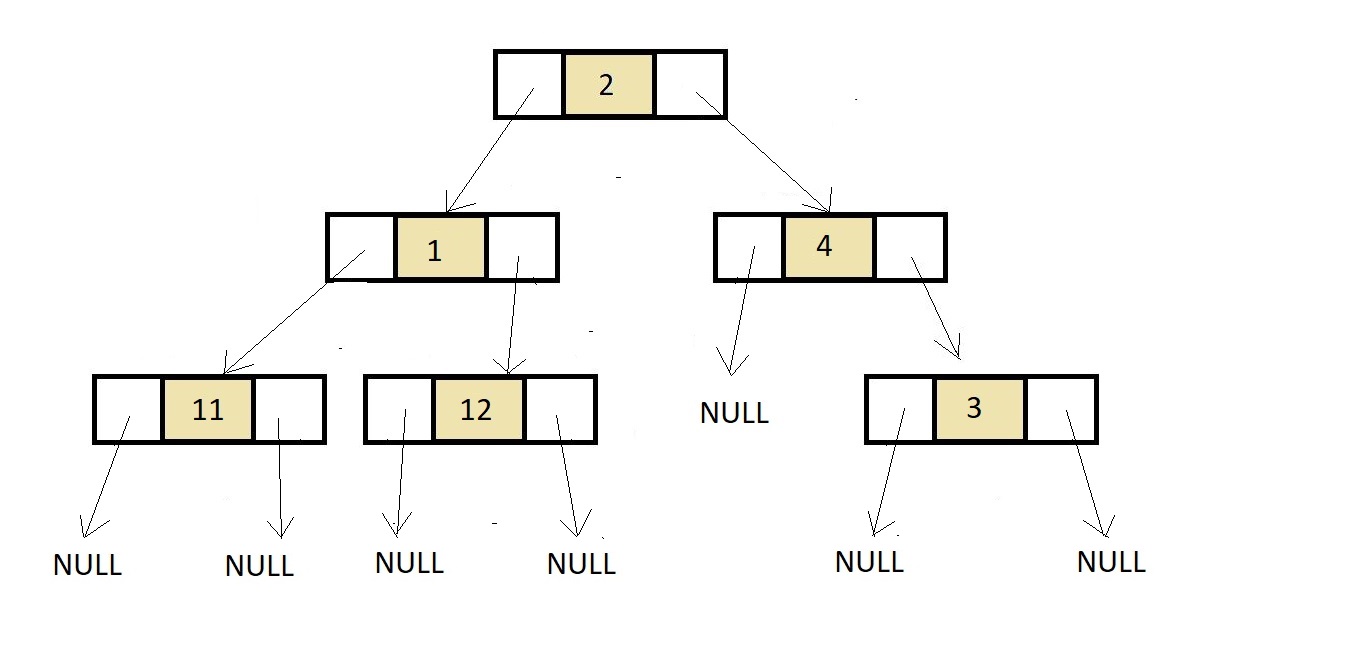
Using links makes the understanding of a binary tree very easy. It actually makes us visualize the tree even. Suppose we have a binary tree of 3 levels.



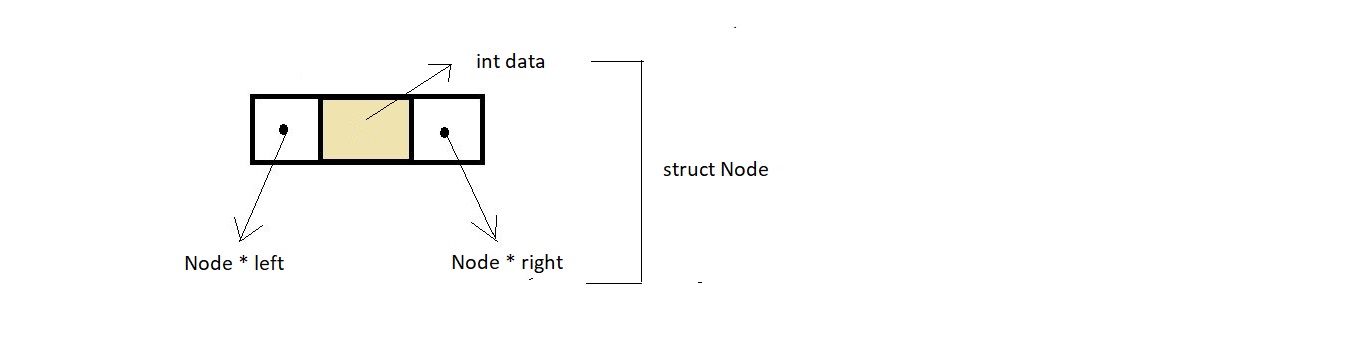
Now if you remember a doubly linked list helped us traversing both to the left and the right. And using that we would create a similar node here, pointing both to the left and the right child node. Follow the below representation of a node here in the linked representation of a binary tree.



You can see how closely this representation resembles a real tree node, unlike the array representation where all the nodes succumbed to a 2D structure.  And now we can very easily transform the whole tree into its linked representation which is just how we imagined it would have looked in real life.



So, this was the representation of the binary tree we saw above using linked representation. And what are these nodes? These are structures having three structure members, first a data element to store the data of the node, and then two structure pointers to hold the address of the child nodes, one for the left, and the other for the right.



**Binary Tree Traversal**

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree −

* In-order Traversal
* Pre-order Traversal
* Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

## In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.



We start from **A**, and following in-order traversal, we move to its left subtree **B**. **B** is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be −

***D → B → E → A → F → C → G***

### Algorithm

Until all nodes are traversed −

**Step 1** − Recursively traverse left subtree.

**Step 2** − Visit root node.

**Step 3** − Recursively traverse right subtree.

## Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**. **B** is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be −

***A → B → D → E → C → F → G***

### Algorithm

Until all nodes are traversed −

**Step 1** − Visit root node.

**Step 2** − Recursively traverse left subtree.

**Step 3** − Recursively traverse right subtree.

## Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.



We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

***D → E → B → F → G → C → A***

### Algorithm

Until all nodes are traversed −

**Step 1** − Recursively traverse left subtree.

**Step 2** − Recursively traverse right subtree.

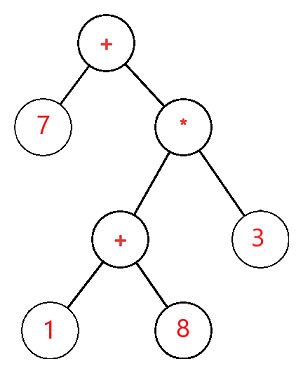
**Step 3** − Visit root node.

# **Expression tree in data structure**

The expression tree is a tree used to represent the various expressions. The tree data structure is used to represent the expressional statements. In this tree, the internal node always denotes the operators.

Expression trees play a very important role in representing the **language-level** code in the form of the data, which is mainly stored in the tree-like structure. It is also used in the memory representation of the **lambda** expression. Using the tree data structure, we can express the lambda expression more transparently and explicitly. It is first created to convert the code segment onto the data segment so that the expression can easily be evaluated.

The expression tree is a binary tree in which each external or leaf node corresponds to the operand and each internal or parent node corresponds to the operators so for example expression tree for 7 + ((1+8)\*3) would be:



**Let S be the expression tree**

If S is not null, then

If S.value is an operand, then

Return S.value

x = solve(S.left)

y = solve(S.right)

Return calculate(x, y, S.value)

Here in the above example, the expression tree used context-free grammar.

We have some productions associated with some production rules in this grammar, mainly known as **semantic rules**. We can define the result-producing from the corresponding production rules using these semantic rules. Here we have used the value parameter, which will calculate the result and return it to the grammar's start symbol. S.left will calculate the left child of the node, and similarly, the right child of the node can be calculated using the S.right parameter.

# **Threaded Binary Tree**

**What do you mean by Threaded Binary Tree?**

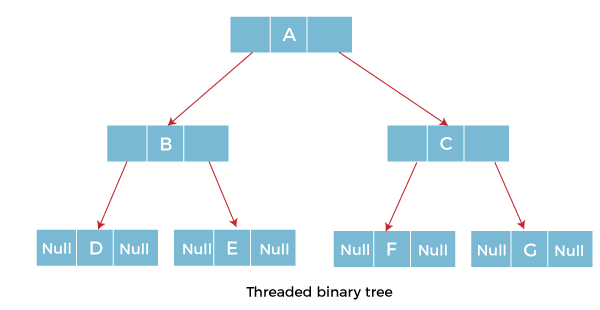
In the linked representation of binary trees, more than one half of the link fields contain NULL values which results in wastage of storage space.

If a binary tree consists of **n** nodes then **n+1** link fields contain NULL values.

So in order to effectively manage the space, a method was devised by Perlis and Thornton in which the NULL links are replaced with special links known as threads.

Such binary trees with threads are known as **threaded binary trees**.

Each node in a threaded binary tree either contains a link to its child node or thread to other nodes in the tree.

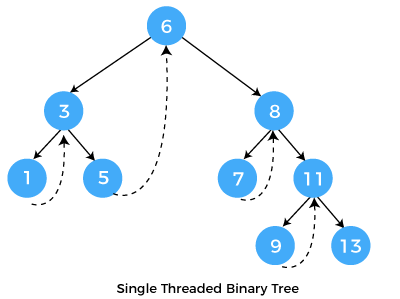


### Types of Threaded Binary Tree

There are two types of threaded Binary Tree:

* One-way threaded Binary Tree
* Two-way threaded Binary Tree

**One-way threaded Binary trees:**



In one-way threaded binary trees, a thread will appear either in the right or left link field of a node.

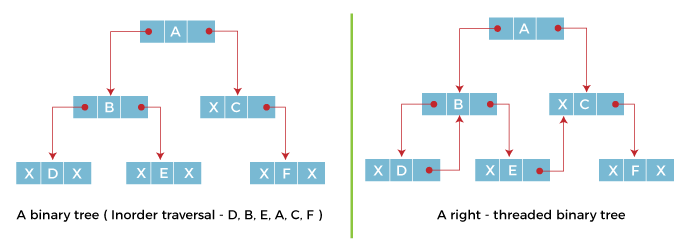
If it appears in the right link field of a node then it will point to the next node that will appear on performing in order traversal.

Such trees are called **Right threaded binary trees**.

If thread appears in the left field of a node then it will point to the nodes inorder predecessor. Such trees are called **Left threaded binary trees.**

Left threaded binary trees are used less often as they don't yield the last advantages of right threaded binary trees.

In one-way threaded binary trees, the right link field of last node and left link field of first node contains a NULL. In order to distinguish threads from normal links they are represented by dotted lines.



The above figure shows the inorder traversal of this binary tree yields D, B, E, A, C, F.

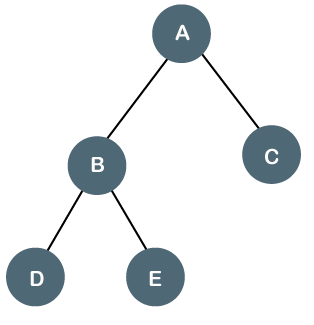
When this tree is represented as a right threaded binary tree, the right link field of leaf node D which contains a NULL value is replaced with a thread that points to node B which is the inorder successor of a node D.

In the same way other nodes containing values in the right link field will contain NULL value.

**1. Full/ proper/ strict Binary tree**

The full binary tree is also known as a strict binary tree. The tree can only be considered as the full binary tree if each node must contain either 0 or 2 children. The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.

**Let's look at the simple example of the Full Binary tree.**



In the above tree, we can observe that each node is either containing zero or two children; therefore, it is a Full Binary tree.

**Properties of Full Binary Tree**

* The number of leaf nodes is equal to the number of internal nodes plus 1. In the above example, the number of internal nodes is 5; therefore, the number of leaf nodes is equal to 6.
* The maximum number of nodes is the same as the number of nodes in the binary tree, i.e., 2h+1 -1.
* The minimum number of nodes in the full binary tree is 2\*h-1.
* The minimum height of the full binary tree is **log2(n+1) - 1.**
* The maximum height of the full binary tree can be computed as:

n= 2\*h - 1

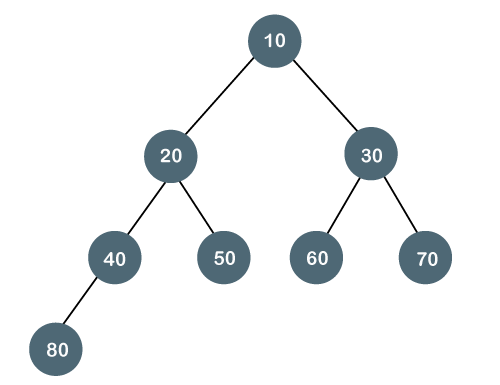
n+1 = 2\*h

**h = n+1/2**

**Complete Binary Tree**

The complete binary tree is a tree in which all the nodes are completely filled except the last level. In the last level, all the nodes must be as left as possible. In a complete binary tree, the nodes should be added from the left.

Let's create a complete binary tree.



The above tree is a complete binary tree because all the nodes are completely filled, and all the nodes in the last level are added at the left first.

**Properties of Complete Binary Tree**

* The maximum number of nodes in complete binary tree is 2h+1 - 1.
* The minimum number of nodes in complete binary tree is 2h.
* The minimum height of a complete binary tree is **log2(n+1) - 1.**
* The maximum height of a complete binary tree is