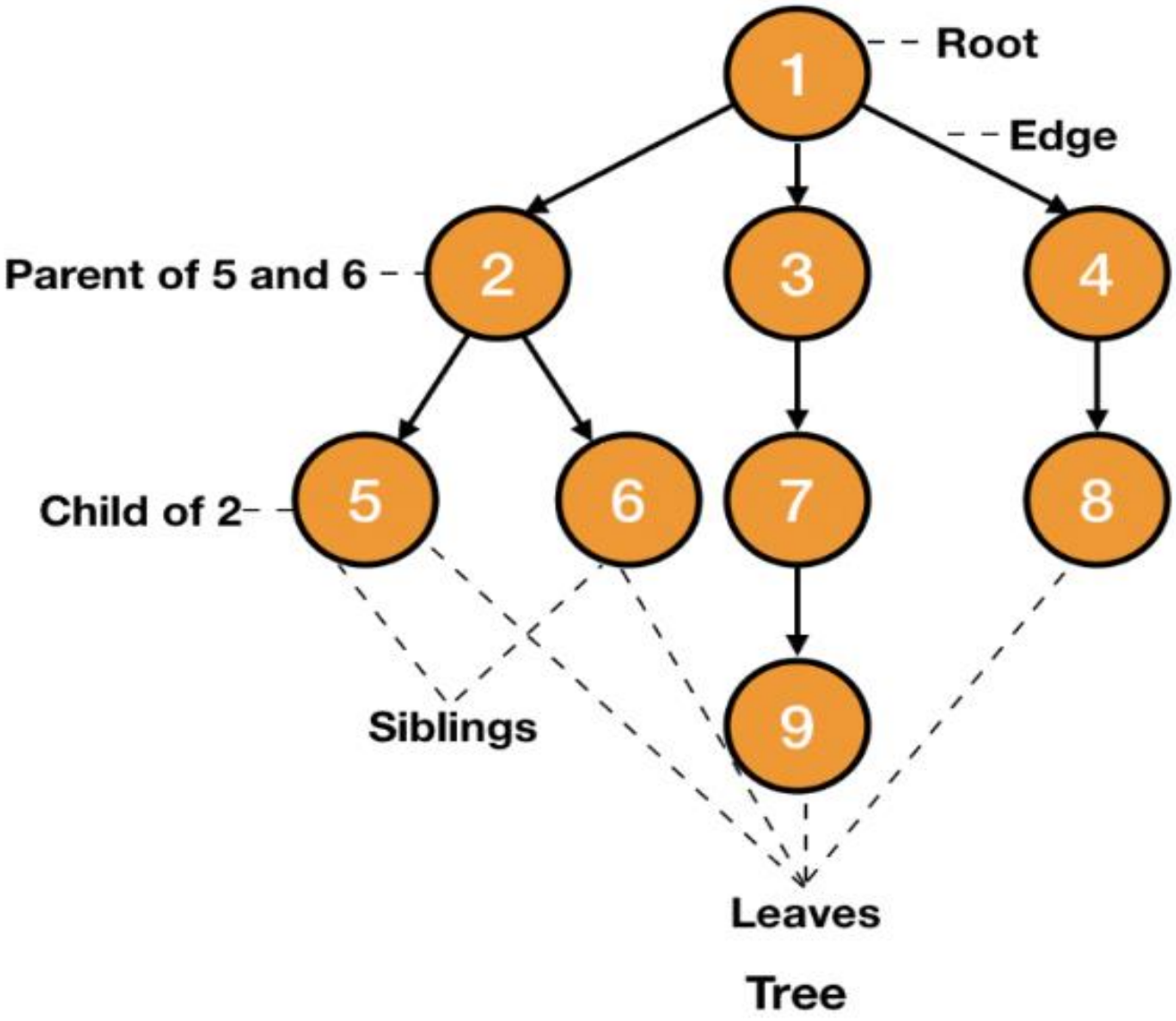
Trees

Productivity experts say that breakthroughs come by thinking "nonlinearly."

Introduction

We are referring to the linear relation by before and after relationship but in case of tree the relation between elements will be above and below.

Tree Definitions and Properties



Terminology	Description	Example from Diagram
Node	Each vertex of a tree is a node.	'1', '2', '3' are the node in tree.
Root	Topmost node of a tree	Node '1' is the topmost root node.
Parent Node	The node having an edge sharing to a child node	Node '2' is the parent of '5' and '6', Node '3' is the parent of '7'.
Child Node	The sub-node of a parent node is the child node	'5' and '6' is the children of '2'.
Leaf	The last node which does have any sub node is the leaf node	'5', '6', '9' and '8' are leaf node.
Edge	Connecting link between two nodes	Link between '1' and '2', '2' and '5', '2' and '6' are edges
Siblings	Nodes with same parent are siblings	'5' and '6' are siblings with '2' as their parent.
Height	Height of a tree is the length of longest path from **root** to **leaf** node. It is calculated with total number of edges	Height of '1' is 3. (Longest path is 1-3-7-9)
Depth	the **number of edges** in the path **from the root** of the tree **to the node**.	Depth of '8' is 2
Level	Each step from top to bottom is called a Level. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on	'1' or root node is at level 0, '2', '3', and '4' is at level 1, and so on.
Sub-Tree	Descendants of a node represent subtree.	Node '2', '5', and '6' represent a sub-tree.
Degree of Node	Degree of a node represents the ***total number of children in it.***	Degree of '2' is 2 ('5' and '6'). Degree of '4' is 1.

Tree Definition

Formally, we define tree T to be a set of nodes storing elements in a parent-child relationship with the following properties:

- If T is non-empty, it has a special node, called the root of T, that has no parent.
- Each node v of T different from the root has a unique parent node w; every node with parent w is a child of w.

Ordered Trees

An ordered tree is an oriented tree in which the children of a node are somehow ordered. It is a rooted tree in which an ordering is specified for the children of each vertex.

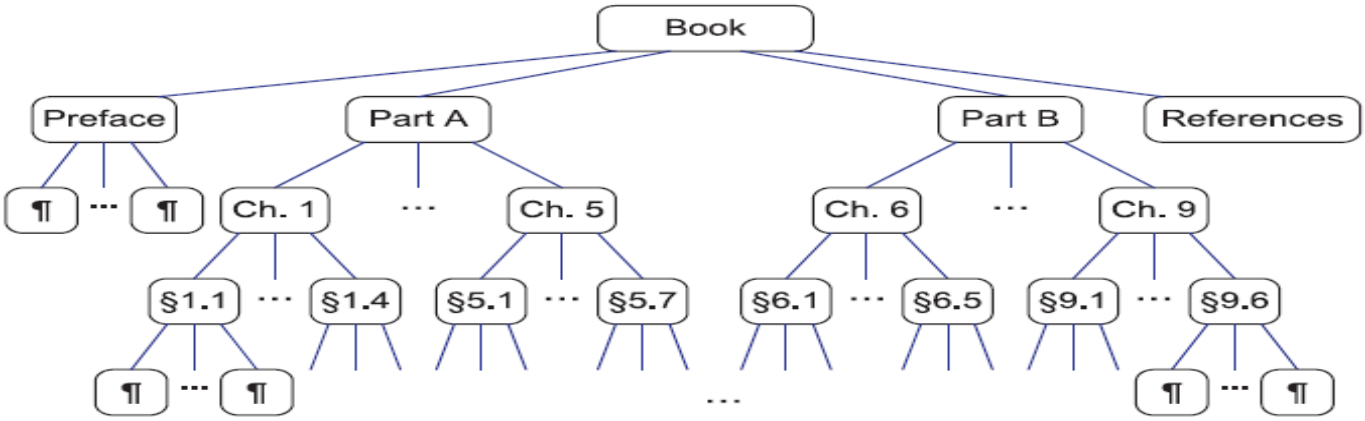


Figure 7.4: An ordered tree associated with a book.

Tree Traversal Algorithms

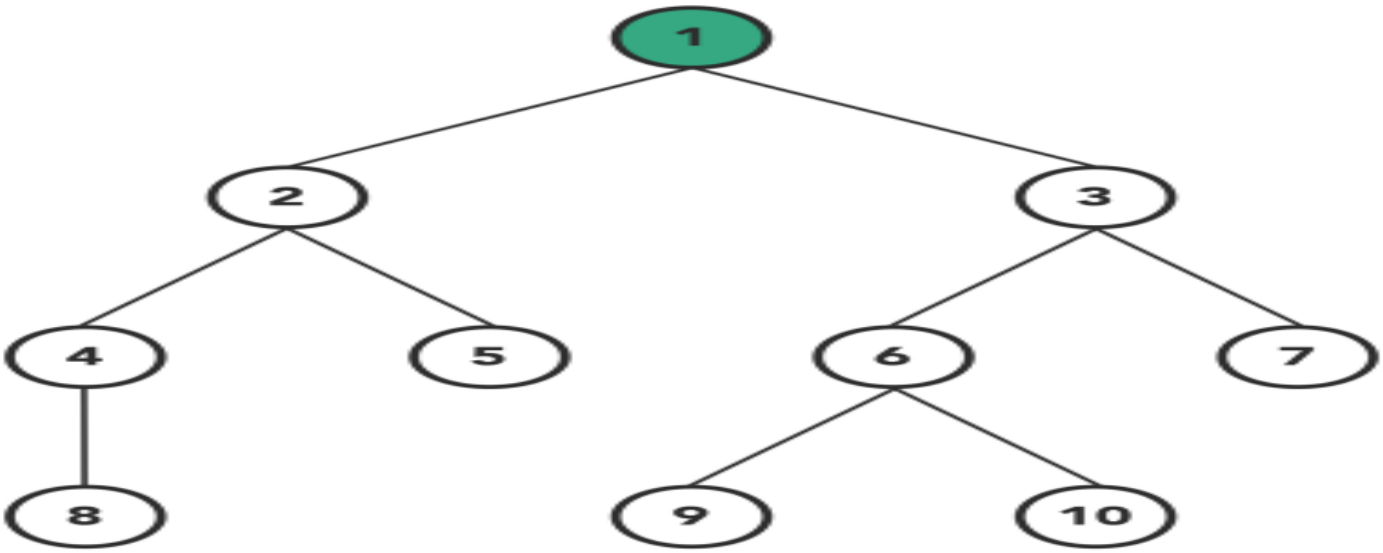
"In computer science, tree traversal (also known as tree search) is a form of graph traversal and refers to the process of visiting (checking and/or updating) each node in a tree data structure, exactly once. Such traversals are classified by the order in which the nodes are visited."

1) Depth-First Search (DFS) Algorithm

1. **Preorder Traversal** (root-left-right)— Visit the current node before visiting any nodes inside left or right subtrees.
2. **In-order Traversal** (left-root-right)— Visit the current node after visiting all nodes inside left subtree but before visiting any node within the right subtree.
3. **Post-order Traversal** (left-right-root) — Visit the current node after visiting all the nodes of left and right subtrees

2) Breadth-First Search (BFS) Algorithm

1. **Level Order Traversal** — Visit nodes level-by-level and left-to-right fashion at the same level



Depth-First Search (DFS)	
Preorder Traversal	1 2 4 8 5 3 6 9 10 7
In-order Traversal	8 4 2 5 1 9 6 10 3 7
Post-order Traversal	8 4 5 2 9 10 6 7 3 1
Breadth-First Search (BFS)	
Level Order Traversal	1 2 3 4 5 6 7 8 9 10

NOTE:

1) All implementations have the same complexity O(n).
2) Post-order Traversal can be used to ***delete*** the tree.
3) Preorder Traversal can be used to ***copy*** the tree.

```
struct node {
    int element;
    struct node* left;
    struct node* right;
};

/*To create a new node*/
struct node* createNode(int val)
{
    struct node* Node = (struct node*)malloc(sizeof(struct node));
    Node->element = val;
    Node->left = NULL;
    Node->right = NULL;
    return (Node);
}
```

```
/*function to traverse the nodes of binary tree in preorder*/
// root left right
void traversePreorder(struct node* root)
{
    if (root == NULL)
        return;
    printf(" %d ", root->element);
    traversePreorder(root->left);
    traversePreorder(root->right);
}
```

```
/*function to traverse the nodes of binary tree in Inorder*/
void traverseInorder(struct node* root)
{
    if (root == NULL)
        return;
    traverseInorder(root->left);
    printf(" %d ", root->element);
    traverseInorder(root->right);
}
```

```
/*function to traverse the nodes of binary tree in postorder*/
void traversePostorder(struct node* root)
{
    if (root == NULL)
        return;
    traversePostorder(root->left);
    traversePostorder(root->right);
    printf(" %d ", root->element);
}
```

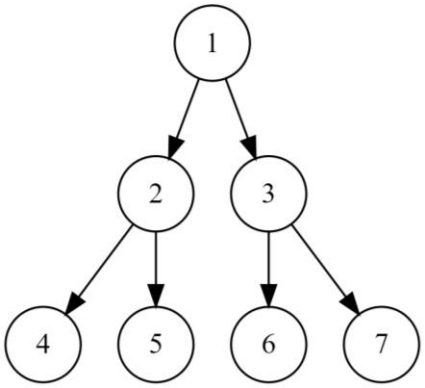
```
int main()
{
    struct node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);

    printf("\n The Preorder traversal of given binary tree is -\n");
    traversePreorder(root);

    printf("\n The Inorder traversal of given binary tree is -\n");
    traverseInorder(root);

    printf("\n The Postorder traversal of given binary tree is -\n");
    traversePostorder(root);

    return 0;
}
```

Types of trees

1) Binary Tree
2) B-Trees
3) Heaps
4) Trees
5) Multiway trees
6) App specific trees

Binary tree

Definition



Binary Tree is defined as a Tree data structure with ***at most two children***. Since each element in a binary tree can have only two children, we typically name them the left and right child.

Binary tree conditions

1) Every node has **at most** two children.
2) Each child node is labeled as being either a left child or a right child.
3) A left child **precedes** a right child in the ordering of children of a node.

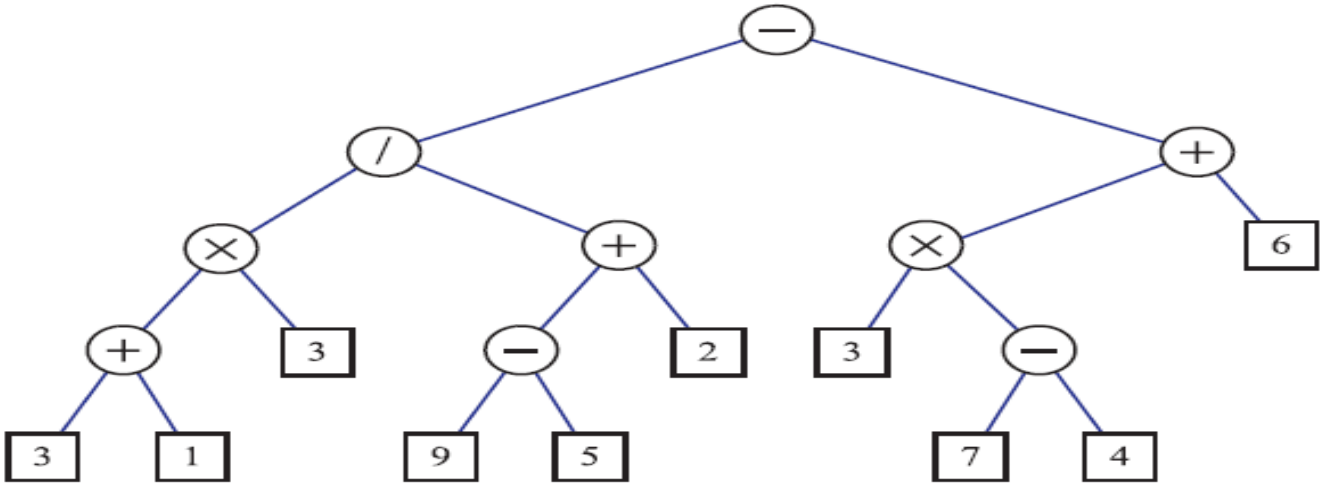
A binary tree is proper if each node has either zero or two children.

Binary Tree Example

An arithmetic expression can be represented by a tree whose external nodes are associated with variables or constants, and whose internal nodes are associated with one of the operators $+ \ - \ * \ /$ Each node in such a tree has a value associated with it.

- If a node is external, then its value is that of its variable or constant.
- If a node is internal, then its value is defined by applying its operation to the values of its children.

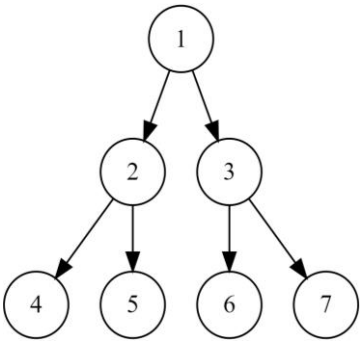
$$\left(\left(\frac{(3+1)\times 3}{(9-5)+2}\right)-\left((3\times(7-4))+6\right)\right)$$

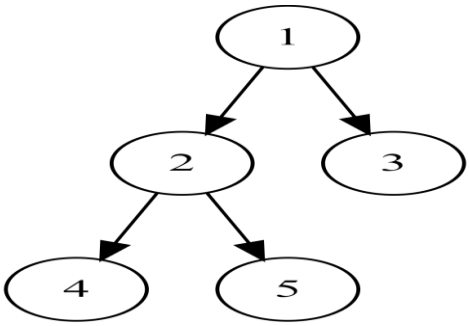


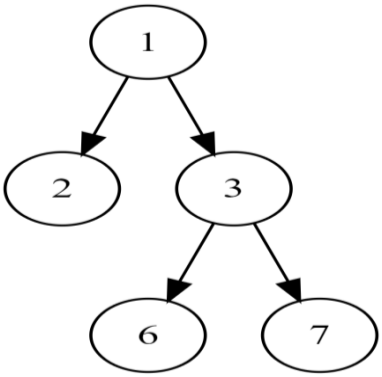
Binary tree types

1) Full binary tree:

A Binary Tree is a full binary tree if every node has 0 or 2 children. We can also say a full binary tree is a binary tree in which all nodes except leaf nodes have two children.



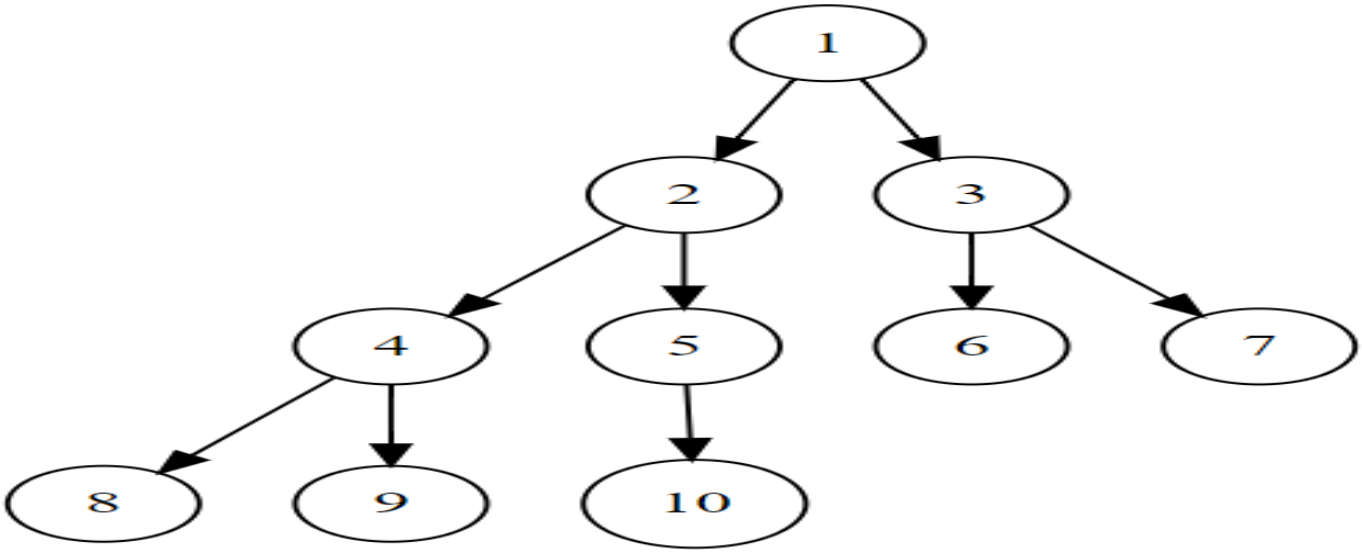




As shown in previous examples each node in the tree has only two children

2) Complete binary tree:

A Binary Tree is a Complete Binary Tree if all the levels are completely filled except possibly the last level and **the last level has all keys as left as possible.**

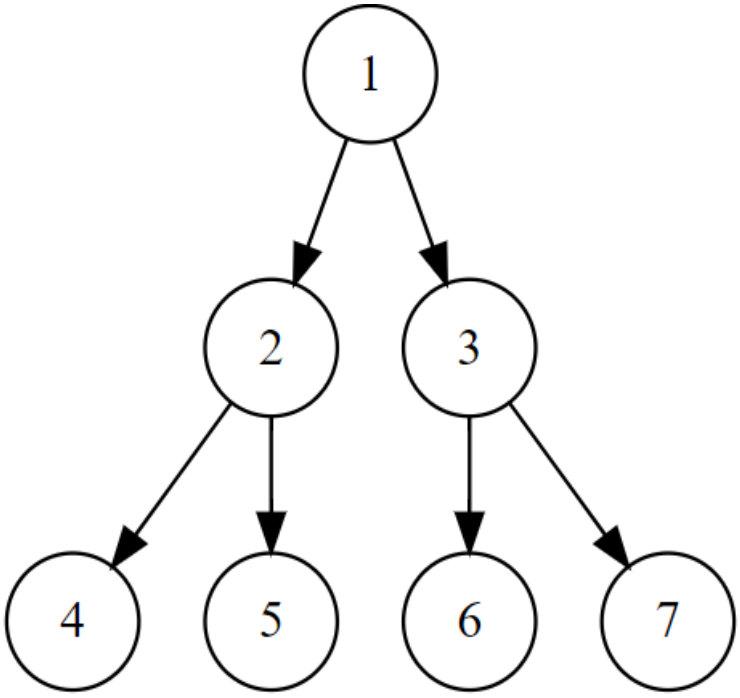


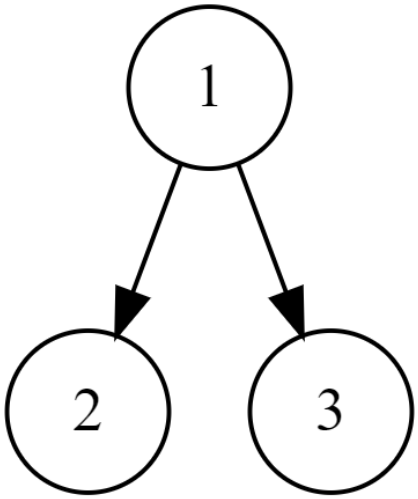
The last leaf element might not have a right sibling i.e., a complete binary tree does not have to be a full binary tree.

Practical example : Binary Heap.

3) Perfect Binary Tree:

A Binary tree is a Perfect Binary Tree in which all the **internal nodes have two children**, and **all leaf nodes are at the same level**.

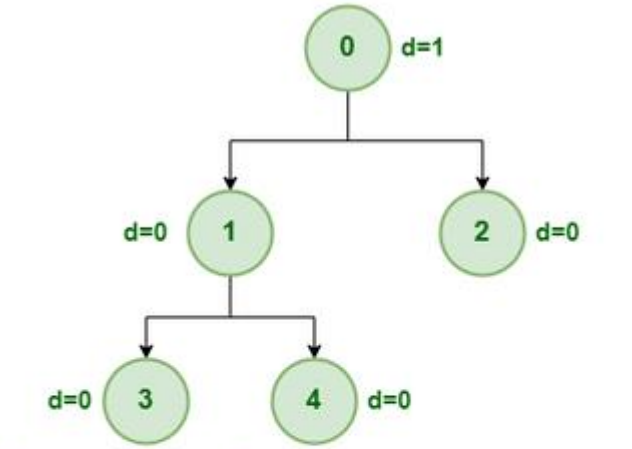




4) Balanced Binary Tree:

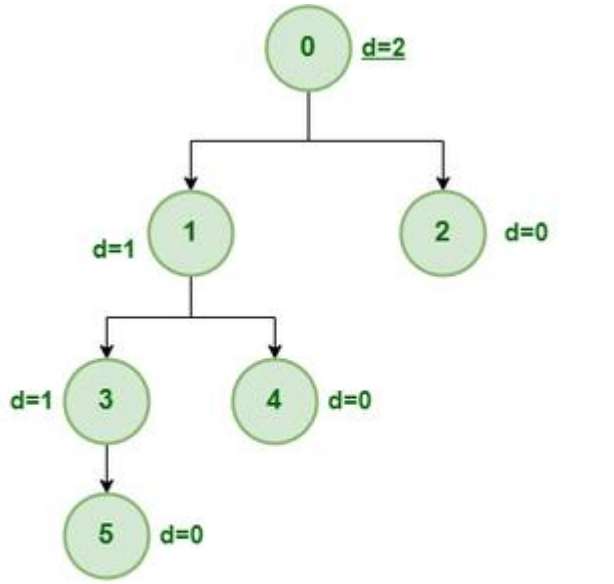
A binary tree is balanced if the height of the tree is $O(\log(n))$ where n is the number of nodes. We can say the tree is balanced if the df is zero or one.

$$df = |\, h(left_{node}) - h(right_{node})| \qquad \{df \leq 1\}$$



Balanced Binary Tree with depth at each level indicated

Depth of a node = |height of left child - height of right child|



Unbalanced Binary Tree with depth at each level indicated

Depth of a node = |height of left child - height of right child|

Examples :

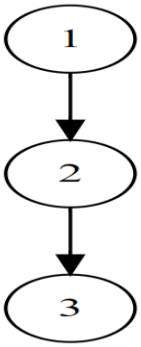
AVLs : the AVL tree maintains $O(\log(n))$ height by making sure that the difference between the heights of the left and right subtrees is at most one.

Red-Black tree : maintain $O(\log(n))$ height by making sure that the number of Black nodes on every root to leaf paths is the same and there are no adjacent red nodes.

Balanced Binary Search trees are performance-wise good as they provide O(log n) time for search, insert, and delete.

5) Degenerate (or pathological) tree:

A Tree where every internal node has one child. Such trees are performance-wise same as linked list.



An external node is one without child branches

Internal node has at least one child branch.

Properties of Binary Trees

Let T be a nonempty binary tree and let $n\ n_e\ n_i\ and\ h$ denote the number of nodes, number of external nodes, number of internal nodes, and height of T, respectively. Then T has the following properties:

1. $h + 1 \leq n \leq 2^{h+1} - 1$	2. $1 \leq n_E \leq 2^h$
3. $h \leq n_I \leq 2^h - 1$	4.$\log(n + 1) - 1 \leq h \leq n - 1$