Tree Terminology

A tree data structure can be defined as follows...

Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.

A tree data structure can also be defined as follows...

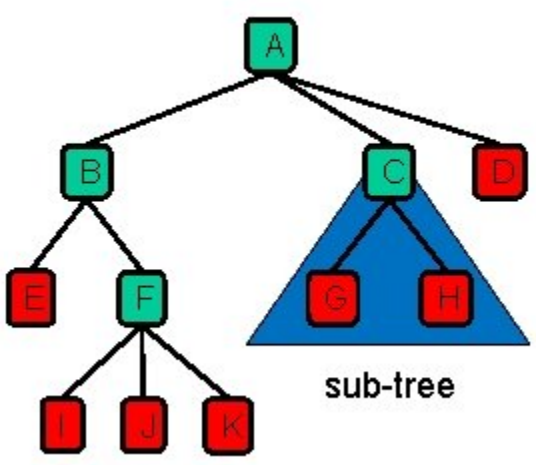
Tree data structure is a collection of data (Node) which is organized in hierarchical structure recursively

In tree data structure, every individual element is called as Node. Node in a tree data structure stores the actual data of that particular element and link to next element in hierarchical structure.

In a tree data structure, if we have N number of nodes then we can have a maximum of N-1 number of links.

Tree Terminology

Example



Root: node without parent (A)

Internal node: node with at least one child (A, B, C, F)

External node (a. k. a. leaf): node without children (E, I, J, K, G, H, D)

Ancestors of a node: parent, grandparent, grand-grandparent, sub-tree etc.

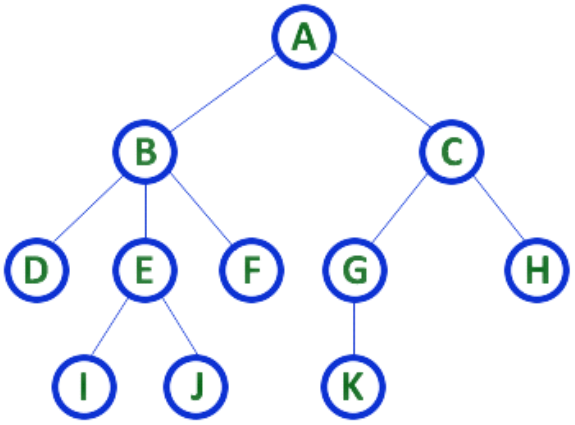
Depth of a node: 8 number of ancestors

Height of a tree: maximum depth of

Descendant of a node: child, grandchild.

Sub-tree: tree consisting of a node and its descendants, any node (3) grand-grandchild, etc.

Consider the following tree...



TREE with 11 nodes and 10 edges

- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

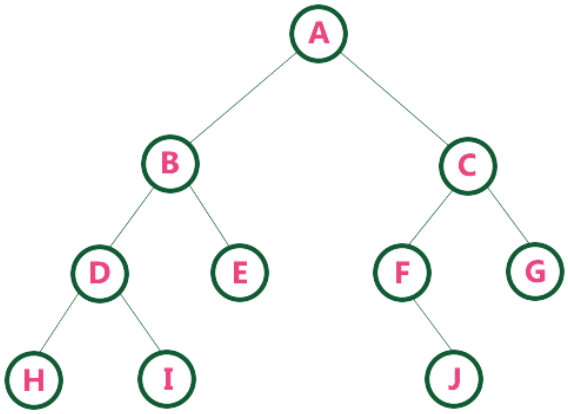
Binary Tree

In a normal tree, every node can have any number of children. A binary tree is a special type of tree data structure in which every node can have a maximum of 2 children. One is known as a left child and the other is known as right child.

A tree in which every node can have a maximum of two children is called Binary Tree.

In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

Example



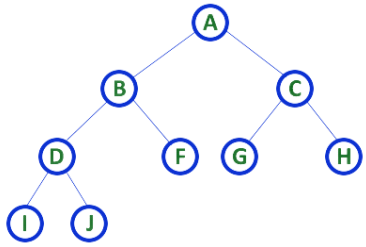
There are different types of binary trees and they are...

1. Strictly Binary Tree:

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none. That means every internal node must have exactly two children. A strictly Binary Tree can be defined as follows...

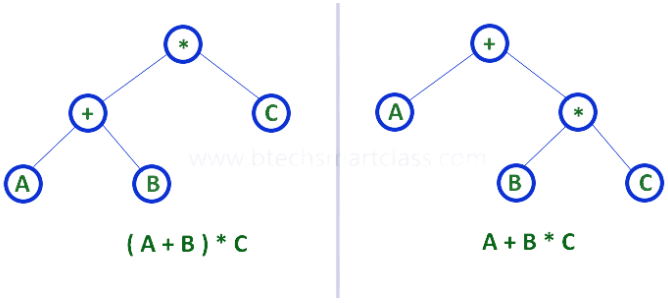
A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree

Strictly binary tree is also called as Full Binary Tree or Proper Binary Tree or 2-Tree



Strictly binary tree data structure is used to represent mathematical expressions.

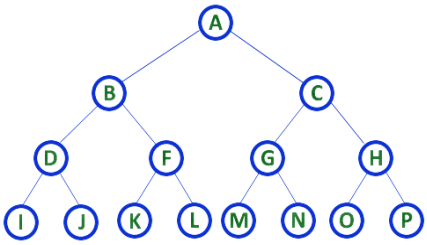
Example



2. Complete Binary Tree:

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none and in complete binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be 2level number of nodes. For example at level 2 there must be 22 = 4 nodes and at level 3 there must be 23 = 8 nodes.

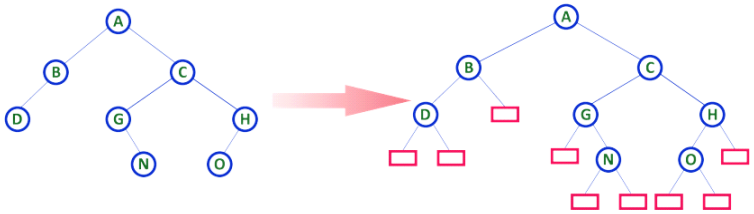
A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree. Complete binary tree is also called as Perfect Binary Tree



3. Extended Binary Tree:

A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required.

The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.



In above figure, a normal binary tree is converted into full binary tree by adding dummy nodes (In pink colour).

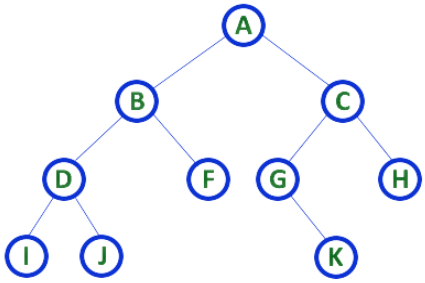
Binary Tree Representations

A binary tree data structure is represented using two methods. Those methods are as follows...

1.Array Representation

2.Linked List Representation

Consider the following binary tree...



1. Array Representation of Binary Tree:

In array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.

Consider the above example of a binary tree and it is represented as follows...

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

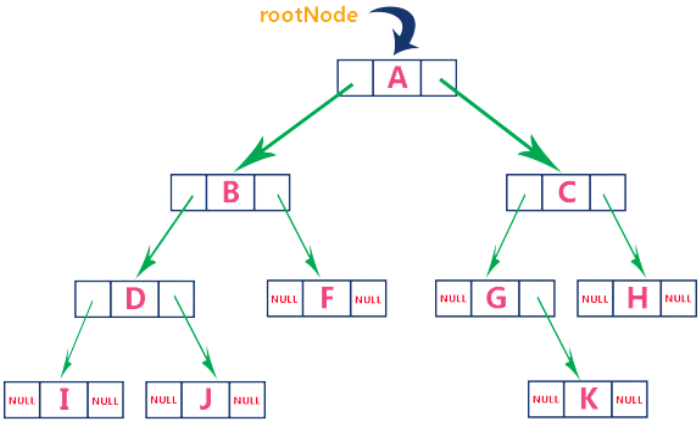
To represent a binary tree of depth 'n' using array representation, we need one dimensional array with a maximum size of 2n + 1.

2. 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...

The above example of the binary tree represented using Linked list representation is shown as follows...



Binary Tree Traversals

A tree traversal visits the nodes of a tree in a systematic manner

Four types of traversal: – –

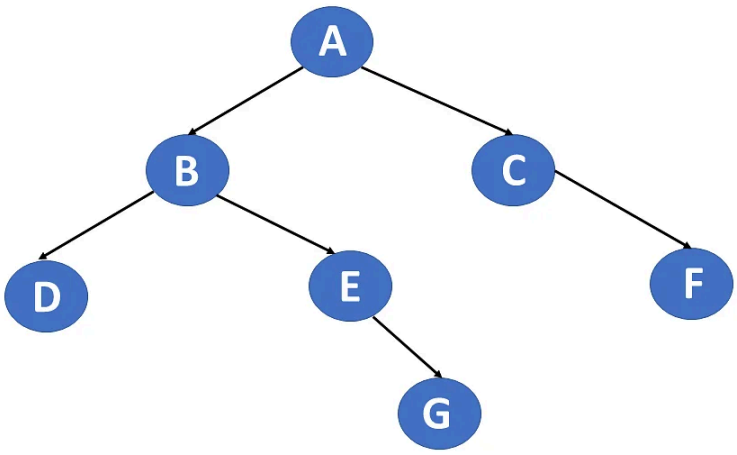
- Pre-order
- Post-order
- In-order

Pre-Order Traversal (Root-Left-Right)

Traverse root node

Traverse left subtree

Traverse right subtree

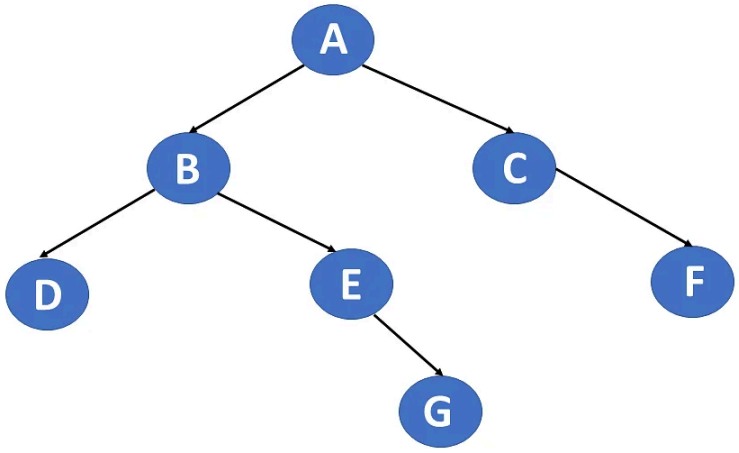


Post-Order Traversal (Left-Right-Root)

Traverse left subtree

Tereveser right subtree

Traverse root node

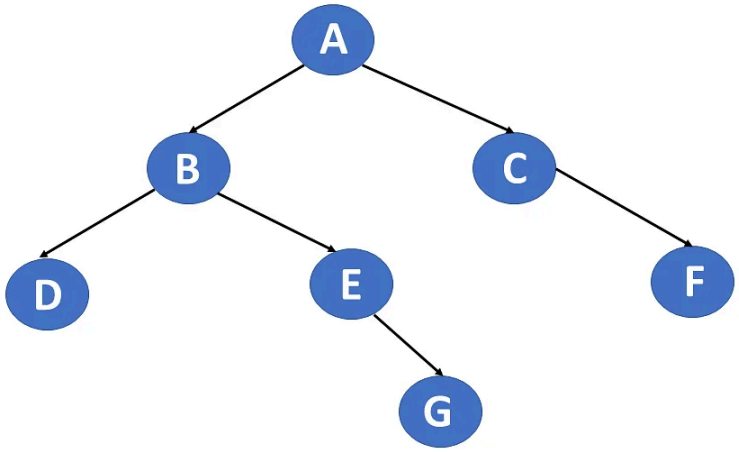


In-Order Traversal (Left-Root-Right)

Traverse left subtree

Traverse root node

Traverse right subtree



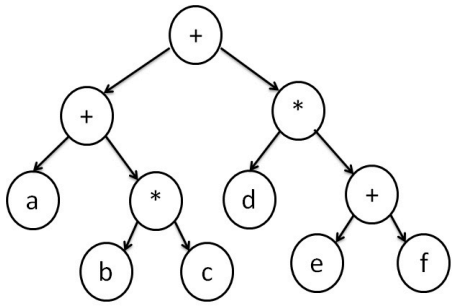
Expression Tree

Expression Tree is used to represent expressions.

An expression and expression tree shown below

a + (b * c) + d * (e + f)

Expression Tree



All the below are also expressions. Expressions may includes constants value as well as variables

1. a * 6
2. 16
3. (a^2)+(b^2)+(2 * a * b)
4. (a/b) + (c)
5. m * (c ^ 2)

It is quite common to use parenthesis in order to ensure correct evaluation of expression as shown above

There are different types of expression formats:

- Prefix expression
- Infix expression and
- Postfix expression

Expression Tree is a special kind of binary tree with the following properties:

- Each leaf is an operand. Examples: a, b, c, 6, 100
- The root and internal nodes are operators. Examples: +, -, *, /, ^
- Subtrees are subexpressions with the root being an operator.

Traversal Techniques

There are 3 standard traversal techniques to represent the 3 different expression formats.

Inorder Traversal

We can produce an infix expression by recursively printing out

- the left expression,
- the root, and
- the right expression.

Postorder Traversal

The postfix expression can be evaluated by recursively printing out

- the left expression,
- the right expression and
- then the root

Preorder Traversal

We can also evaluate prefix expression by recursively printing out:

- the root,
- the left expressoion and
- the right expression.

If we apply all these strategies to the sample tree above, the outputs are:

- Infix expression:
  (a+(b*c))+(d*(e + f))
- Postfix Expression:
  a b c * + d e f + * +
- Prefix Expression:
  + + a * b c * d + e f

Construction of Expression Tree

Let us consider a postfix expression is given as an input for constructing an expression tree. Following are the step to construct an expression tree:

- Read one symbol at a time from the postfix expression.
- Check if the symbol is an operand or operator.
- If the symbol is an operand, create a one node tree and push a pointer onto a stack

- If the symbol is an operator, pop two pointers from the stack namely T1 & T2 and form a new tree with root as the operator, T1 & T2 as a left and right child
- A pointer to this new tree is pushed onto the stack

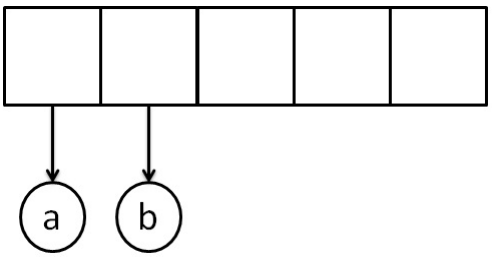
Thus, An expression is created or constructed by reading the symbols or numbers from the left. If operand, create a node. If operator, create a tree with operator as root and two pointers to left and right subtree

Example - Postfix Expression Construction

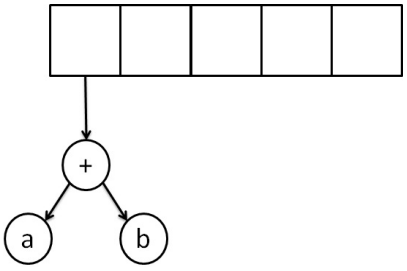
The input is:

a b + c *

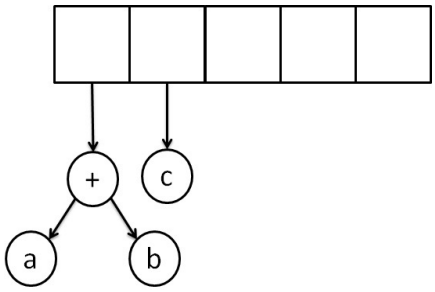
The first two symbols are operands, we create one-node tree and push a pointer to them onto the stack.



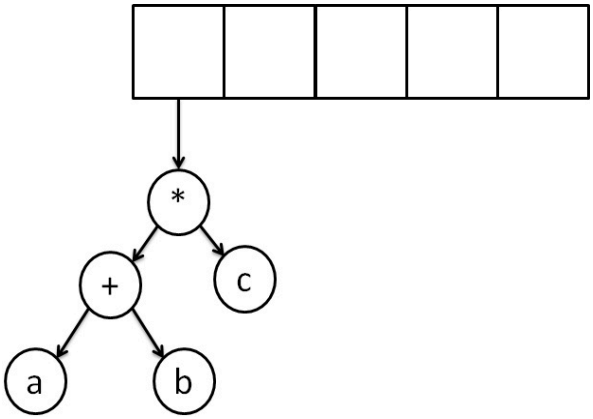
Next, read a'+' symbol, so two pointers to tree are popped,a new tree is formed and push a pointer to it onto the stack.



Next, 'c' is read, we create one node tree and push a pointer to it onto the stack.



Finally, the last symbol is read ' * ', we pop two tree pointers and form a new tree with a, ' * ' as root, and a pointer to the final tree remains on the stack.

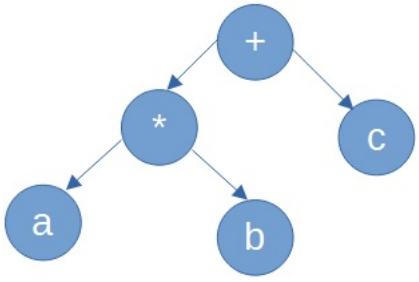


Expression Tree - Examples

This page is specific for Examples of Expression Trees along with expressions. To learn about Expression Tree Traversals, please click on links above.

Expression Tree is used to represent expressions. Let us look at some examples of prefix, infix and postfix expressions from expression tree for 3 of the expresssions:

- a*b+c
- a+b*c+d
- a+b-c*d+e*f



Expression Tree for a*b+c

Expressions from Expression Tree

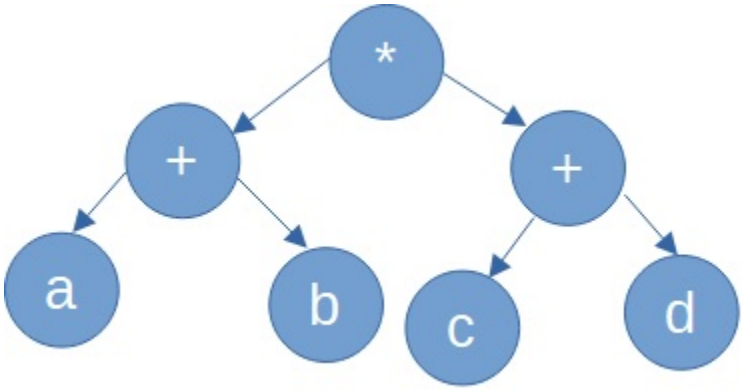
Infix expression a * b + c

Prefix expression + * a b c

Postfix expression a b * c +

Infix, Prefix and Postfix Expressions from Expression Tree for a+b*c+d

Expression Tree for a + b * c + d can be represented as:



Expression Tree for a + b * c + d

Expressions for the binary expression tree above can be written as

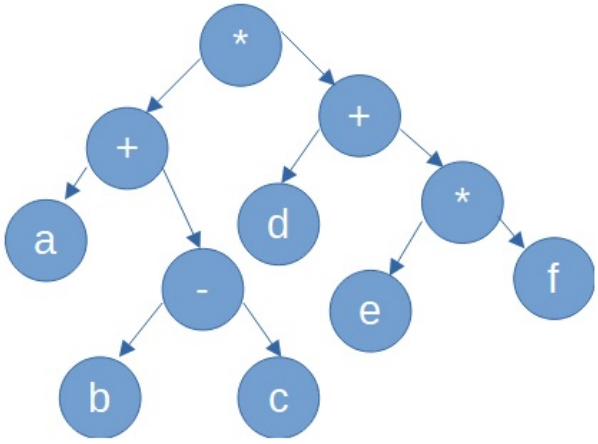
Infix expression a + b * c + d

Prefix expression * + a b + c d

Postfix expression a b + c d + *

Infix, Prefix and Postfix Expressions from Expression Tree for a+b-c*d+e*f

Expression Tree for a + b - c * d + e * f can be represented as:



Expression Tree for a+b-c*d+e*f

Expressions for the binary expression tree above can be written as

Infix expression a + b - c *d + e * f

Prefix expression * + a - b c + d * e f

Postfix expression a b c - + d e f * + *

Binary Search Tree

Binary Search Tree provides a data structure with efficient insertion, deletion and search capability. Binary Search Tree is a binary tree with the following properties:

- All items in the left subtree are less than the root.
- All items in the right subtree are greater than or equal to root.
- Each subtree is itself a binary search tree

Binary Search Tree - Operations

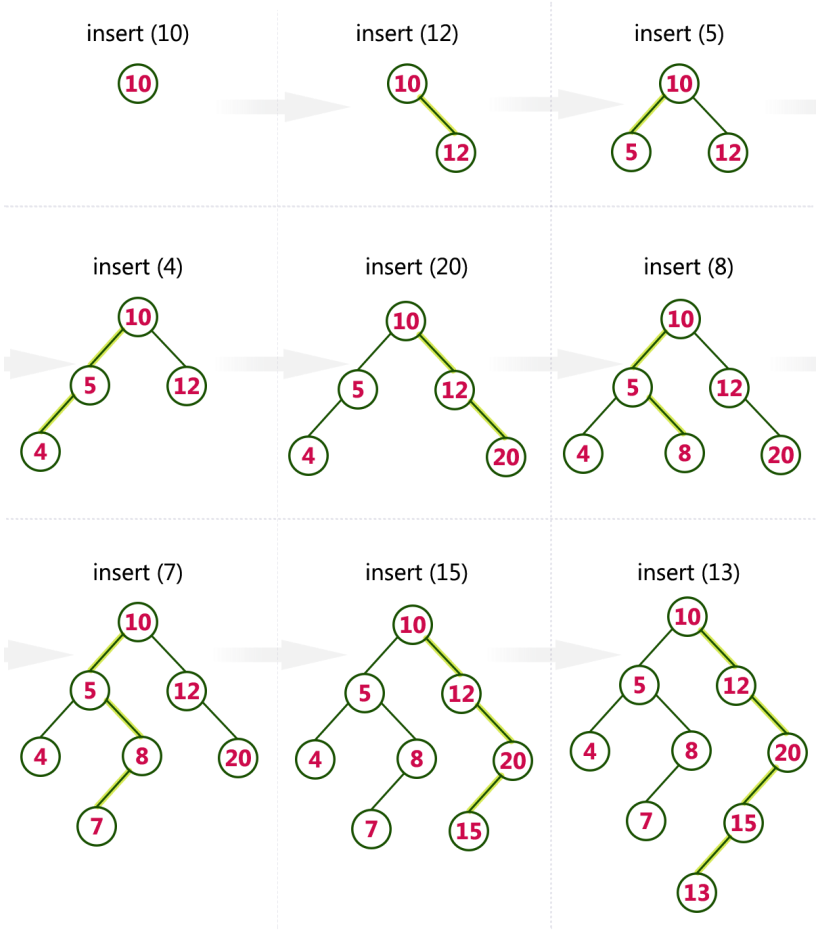
- Search: check the key in a tree.
- Insertion: insert a new element into the tree.
- FindMin: find the minimum element in the tree.
- FindMax: find the maximum element in the tree.
- Deletion: remove an element from the tree.
- Traversal: Visiting every node in the tree

Example

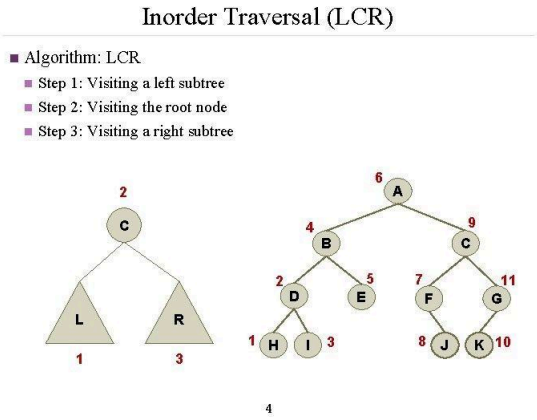
Construct a Binary Search Tree by inserting the following sequence of numbers...

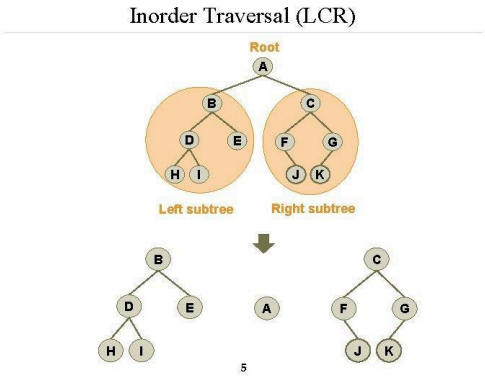
10,12,5,4,20,8,7,15 and 13

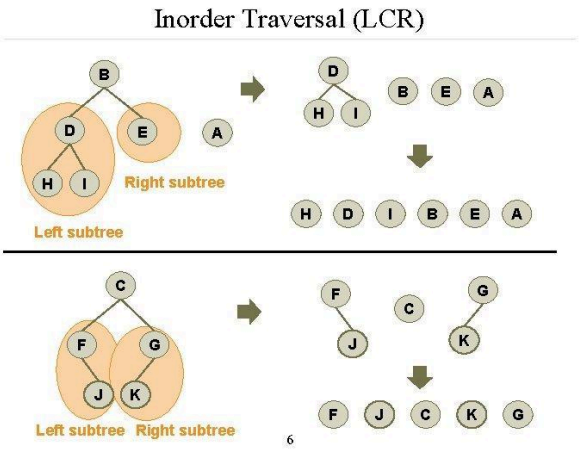
Above elements are inserted into a Binary Search Tree as follows...

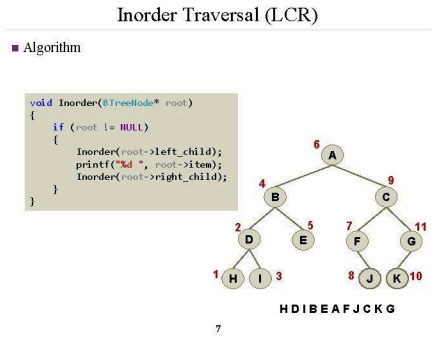


Tree to Inorder Traversal:





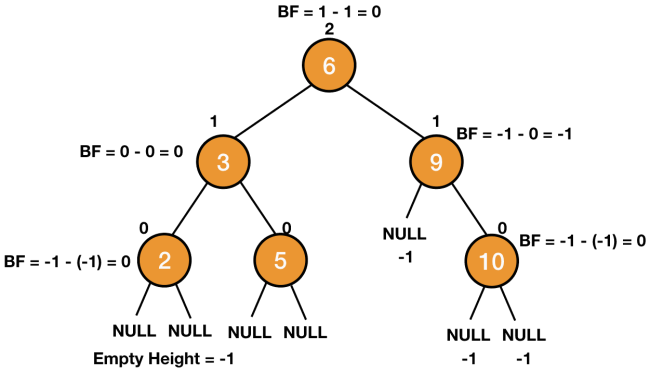




AVL Trees

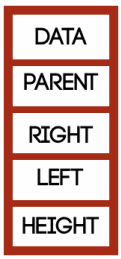
Similar to red-black trees, **AVL (Adelson-Velskii and Landis)** trees are also binary search trees which are pretty good balanced. AVL trees use different rules to achieve that balance.

AVL trees use **balance factor** to get a balanced tree. **Balance factor** of any node is defined as **height(left subtree) - height(right subtree)**.

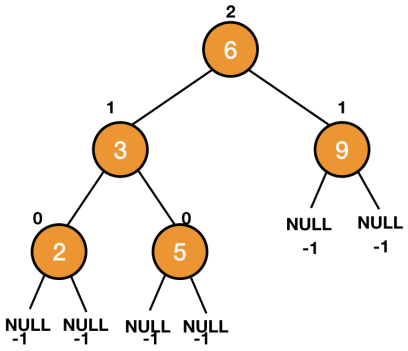


For an AVL tree, the absolute value of balance factor for any node can't be greater than 1 i.e., each node must have a balance factor of either -1, 0 or 1.

Instead of calculating heights of nodes, again and again, we store the current heights in each node. Look at the following picture.

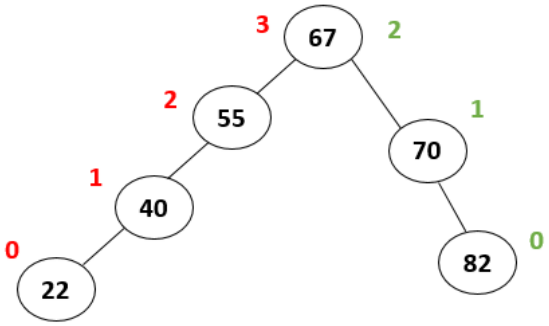


If there is only one child of a node, then the height of the other child (empty height) is assumed to be -1. It is shown in the picture given below.



Why do we need an AVL tree in DS?

To resolve such issues and decrease the searching time, AVL trees were invented by Adelson, Velski & Landis.



- In the above figure, Height of left subtree = 3 was as

Height of right subtree = 0

- Thus Balance Factor = 3-0 = 3 . Thus searching for an element in such a tree has O(n) of complexity which is similar to linear search. To avoid that complex search AVL tree was introduced where every node in the tree needs to maintain balance factor <=1, otherwise various rotation techniques are to be performed to balance such tree.

Types of Rotations

When the tree's balance factor does not satisfy <=1 condition, then rotations need to be performed on them to turn it into a balanced tree.

There are 4 types of rotations:

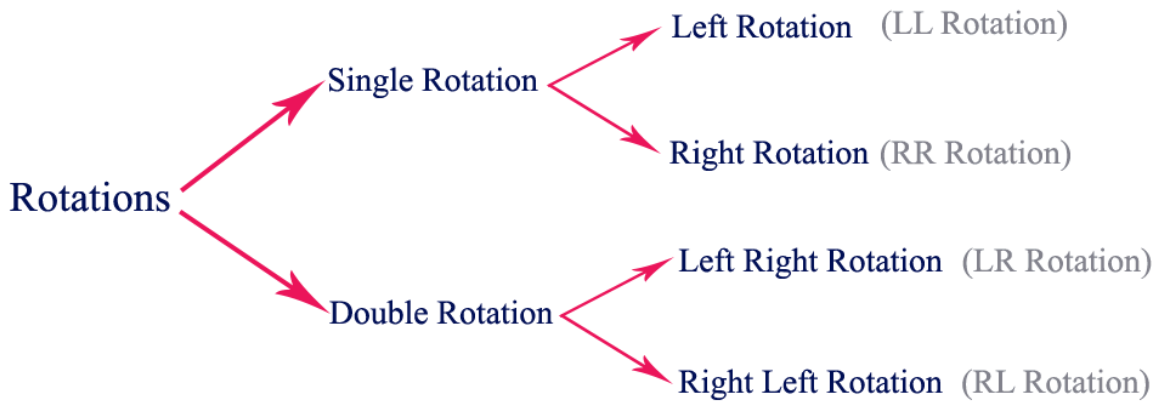
1. Left Rotation: If the addition of a node to the tree's right makes it imbalance then, in that case, Left Rotation needs to be performed.

2. Right Rotation: If the addition of a node to the left of the tree makes the node imbalance then Right Rotation needs to be performed. In other words, When the number of nodes increases on the left side, then there emerges a need to shift the elements to the right side to balance it thus it is said to be Right Rotation.

3. Left-Right Rotation: This type of rotation is a combination of the above 2 rotations explained. This type of rotation occurs when one element is added to the right subtree of a left tree.

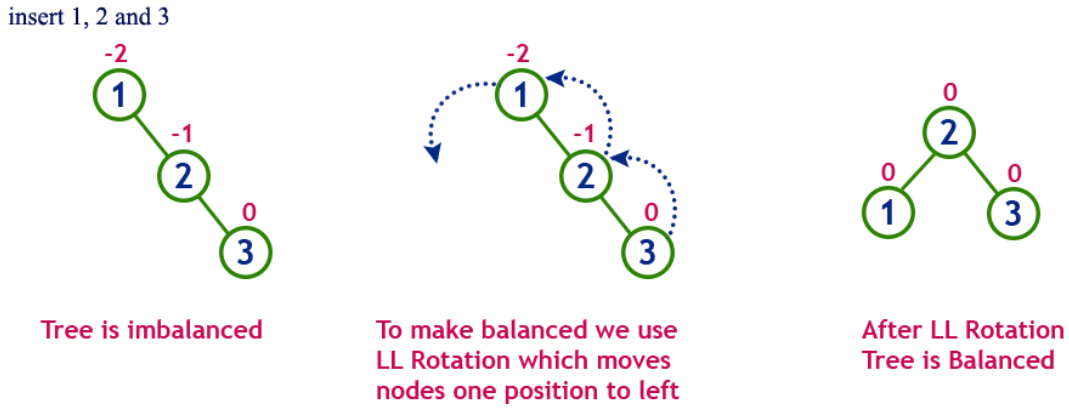
In such a case first, perform left rotation on the subtree followed by a right rotation of the left tree.

4. Right-Left Rotation: This type of rotation is also composed of a sequence of above 2 rotations. This type of rotation is needed when an element is added to the left of the right subtree, and the tree becomes imbalanced. In such a case, we first perform right rotation on the right subtree and then left rotation on the right tree.



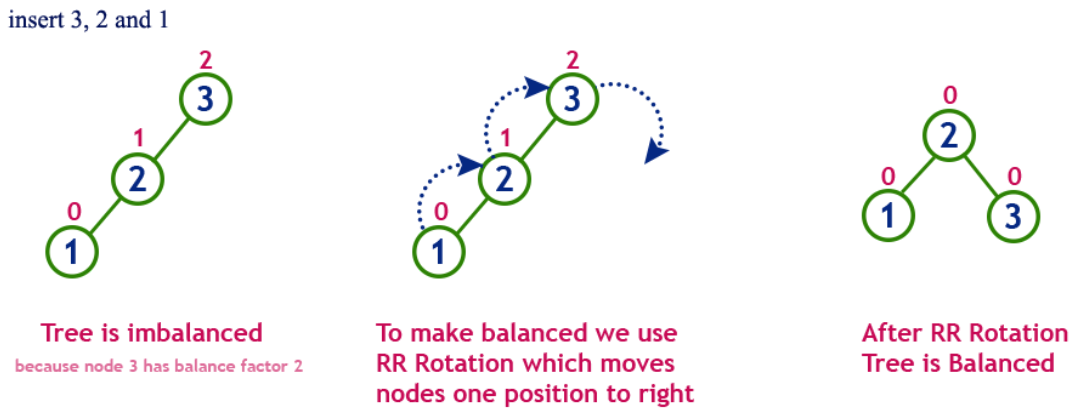
Single Left Rotation (LL Rotation)

In LL Rotation, every node moves one position to left from the current position. To understand LL Rotation, let us consider the following insertion operation in AVL Tree...



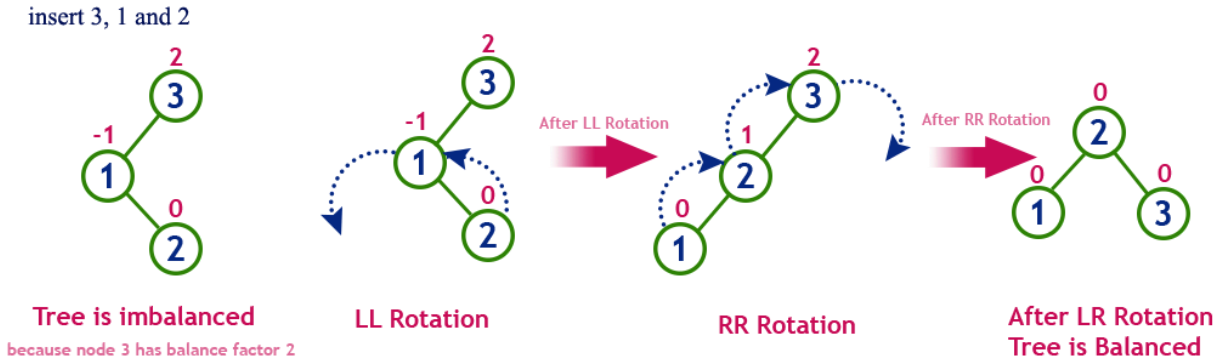
Single Right Rotation (RR Rotation)

In RR Rotation, every node moves one position to right from the current position. To understand RR Rotation, let us consider the following insertion operation in AVL Tree...



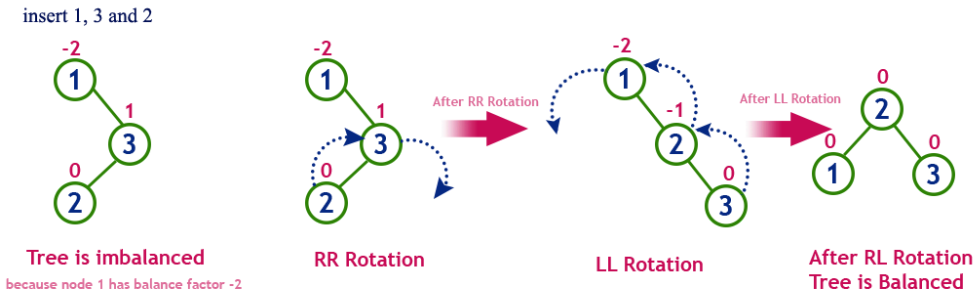
Left Right Rotation (LR Rotation)

The LR Rotation is a sequence of single left rotation followed by a single right rotation. In LR Rotation, at first, every node moves one position to the left and one position to right from the current position. To understand LR Rotation, let us consider the following insertion operation in AVL Tree...



Right Left Rotation (RL Rotation)

The RL Rotation is sequence of single right rotation followed by single left rotation. In RL Rotation, at first every node moves one position to right and one position to left from the current position. To understand RL Rotation, let us consider the following insertion operation in AVL Tree...



Applications of binary trees

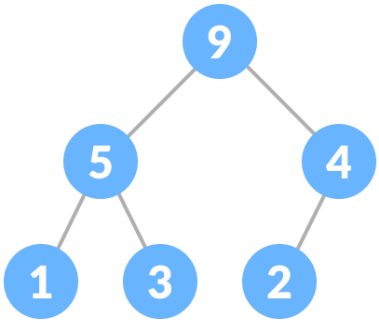
- Binary Search Tree - Used in *many* search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
- Binary Space Partition - Used in almost every 3D video game to determine what objects need to be rendered.
- Binary Tries - Used in almost every high-bandwidth router for storing router-tables.

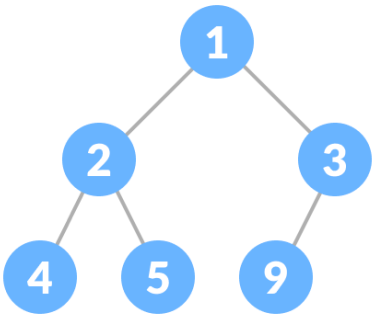
- Hash Trees - Used in torrents and specialized image-signatures in which a hash needs to be verified, but the whole file is not available. Also used in blockchains for eg. Bitcoin.
- Heaps - Used in implementing efficient priority-queues, which in turn are used for scheduling processes in many operating systems, Quality-of-Service in routers, and A* *(path-finding algorithm used in AI applications, including robotics and video games)*. Also used in heap-sort.
- Huffman Coding Tree (Chip Uni) - Used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats.
- GGM Trees - Used in cryptographic applications to generate a tree of pseudo-random numbers.
- Syntax Tree - Constructed by compilers and (implicitly) calculators to parse expressions.
- Treap - Randomized data structure used in wireless networking and memory allocation.
- T-tree - Though most databases use some form of B-tree to store data on the drive, databases which keep all (most) their data in memory often use T-trees to do so.

Heap Tree:

Heap data structure is a complete binary tree that satisfies the heap property, where any given node is

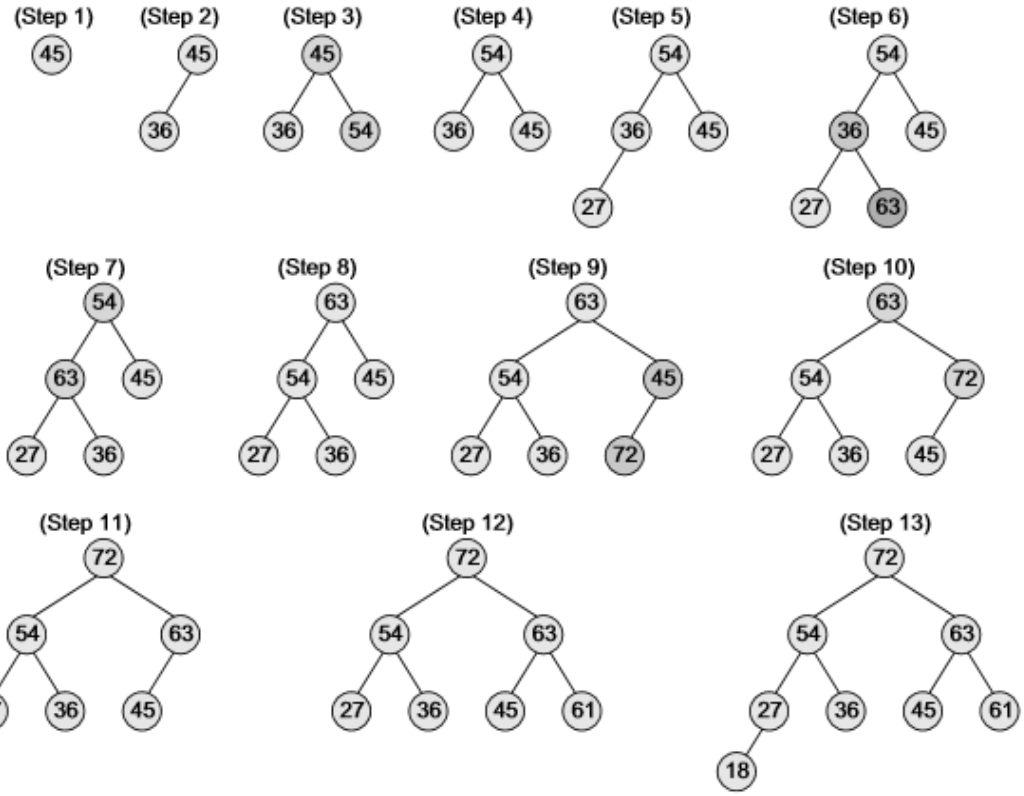
- always greater than its child node/s and the key of the root node is the largest among all other nodes. This property is also called max heap property.
- always smaller than the child node/s and the key of the root node is the smallest among all other nodes. This property is also called min heap property.





Build a max heap H from the given set of numbers: 45, 36, 54, 27, 63, 72, 61,and 18. Also draw the memory representation of the heap.

SOLUTION :-



Threaded Binary Tree

Threaded Binary Tree is also a binary tree in which all left child pointers that are NULL (in Linked list representation) points to its in-order predecessor, and all right child pointers that are NULL (in Linked list representation) points to its in-order successor

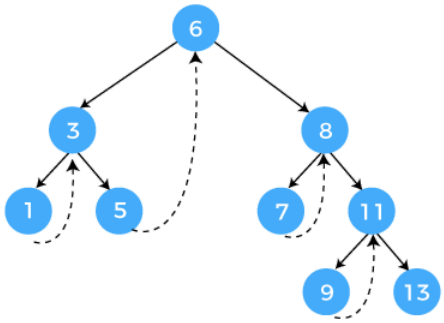
In any binary tree linked list representation, there is a number of NULL pointers than actual pointers. Generally, in any binary tree linked list representation, if there are **2N** number of reference fields, then **N+1** number of reference fields are filled with NULL (**N+1 are NULL out of 2N**). This NULL pointer does not play any role except indicating that there is no link (no child).

Types of Threaded Binary Tree

There are two types of threaded Binary Tree:

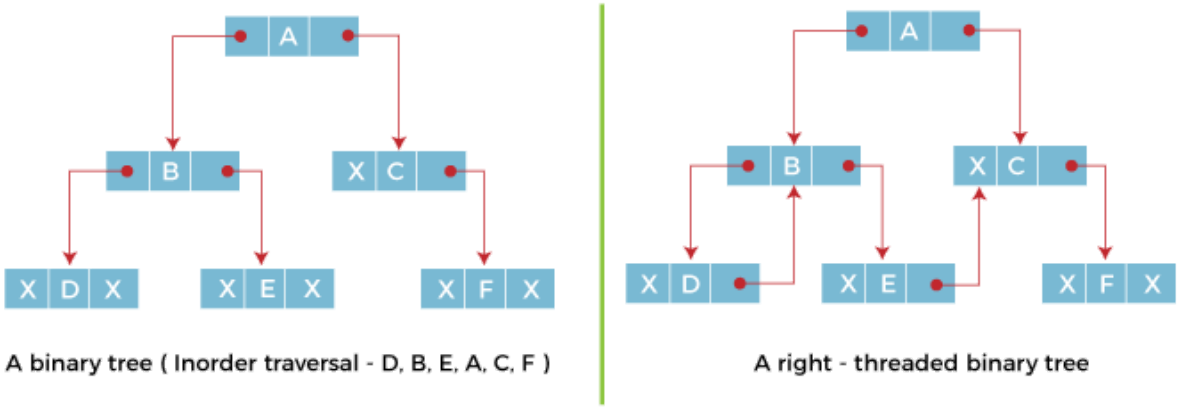
- One-way threaded Binary Tree
- Two-way threaded Binary Tree

One-way threaded Binary trees:

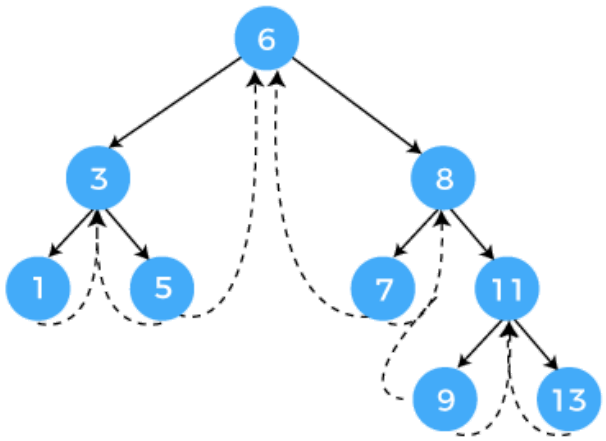


Single Threaded Binary Tree

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.

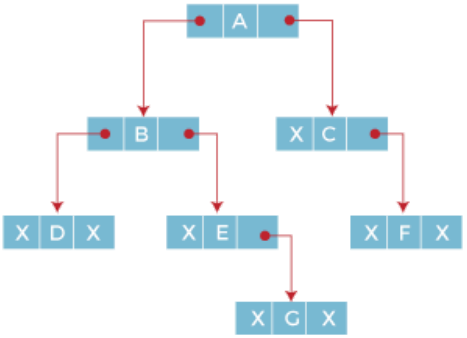


The above figure shows the inorder traversal of this binary tree yields D, B, E, A, C, F. When this tree is represented as a right threaded binary tree, the right link field of leaf node D which contains a NULL value is replaced with a thread that points to node B which is the inorder successor of a node D. In the same way other nodes containing values in the right link field will contain NULL value.

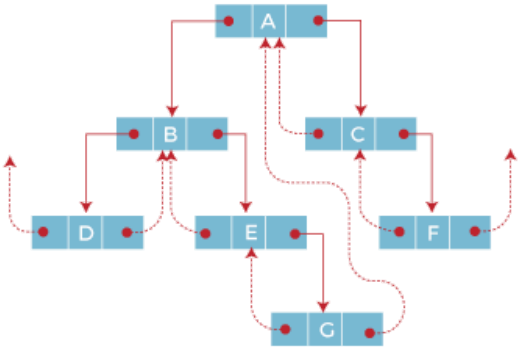


Double Threaded Binary Tree

In two-way threaded Binary trees, the right link field of a node containing NULL values is replaced by a thread that points to nodes inorder successor and left field of a node containing NULL values is replaced by a thread that points to nodes inorder predecessor.

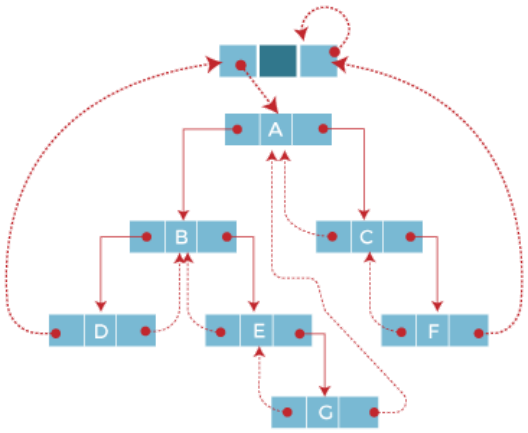


A binary tree (Inorder traversal - D, B, E, G, A, C, F)



A two - way threaded binary tree

The above figure shows the inorder traversal of this binary tree yields D, B, E, G, A, C, F. If we consider the two-way threaded Binary tree, the node E whose left field contains NULL is replaced by a thread pointing to its inorder predecessor i.e. node B. Similarly, for node G whose right and left linked fields contain NULL values are replaced by threads such that right link field points to its inorder successor and left link field points to its inorder predecessor. In the same way, other nodes containing NULL values in their link fields are filled with threads.



Two-way threaded - tree with header node