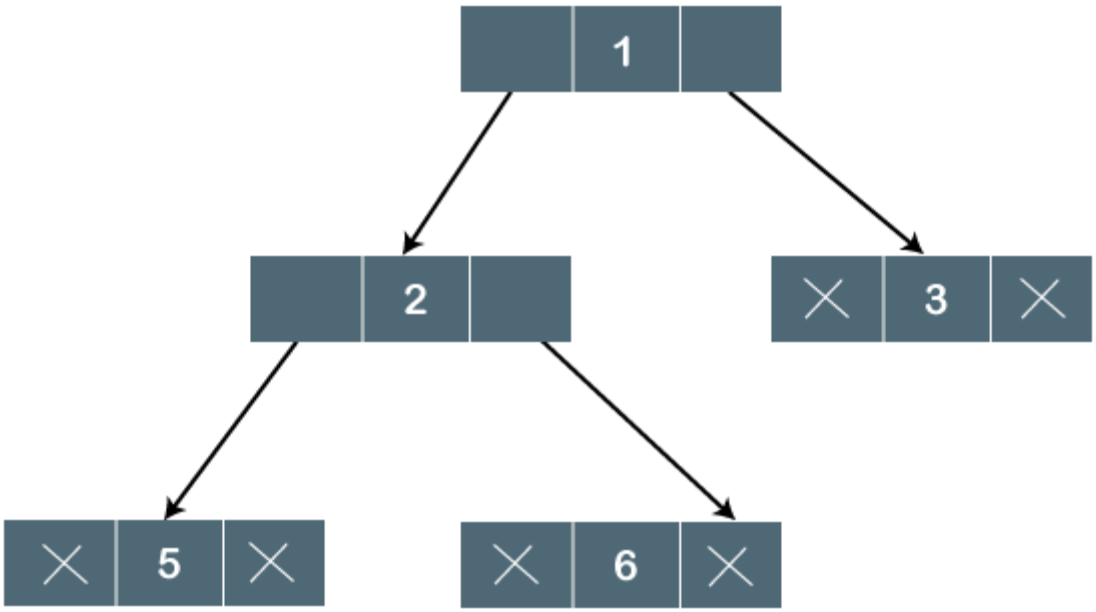
Binary Tree:

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.

Properties of Binary Tree

- At each level of i, the maximum number of nodes is 2^i.
- 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 $(2^0 + 2^1 + 2^2 + \ldots 2^h) = 2^{h+1} - 1$.

- 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.

If there are 'n' number of nodes in the binary tree.

The minimum height can be computed as:

As we know that,

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

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

Taking log on both the sides,

$\log_2(n+1) = \log 2(2^{h+1})$

$\log_2(n+1) = h+1$

$h = \log_2(n+1) - 1$

The maximum height can be computed as:

As we know that,

$n = h+1$

$h = n-1$

Types of Binary Tree

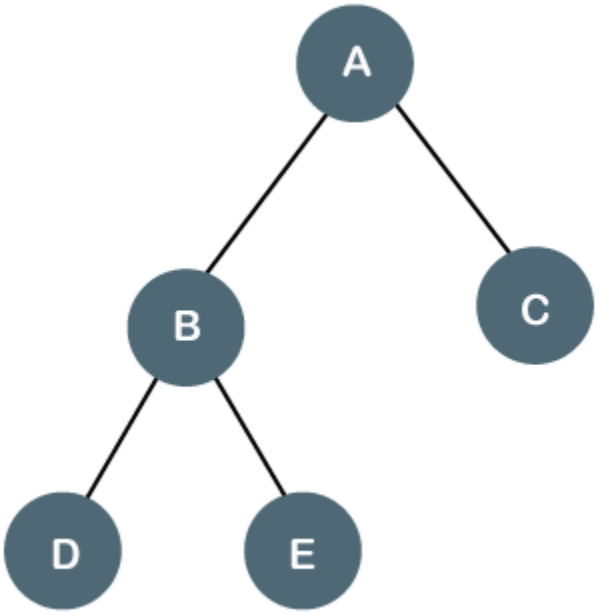
There are four types of Binary tree:

- Full/ proper/ strict Binary tree
- Complete Binary tree
- Perfect Binary tree
- Degenerate Binary tree
- Balanced Binary tree

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., 2^{h+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 **$\log_2(n+1)$ - 1.**

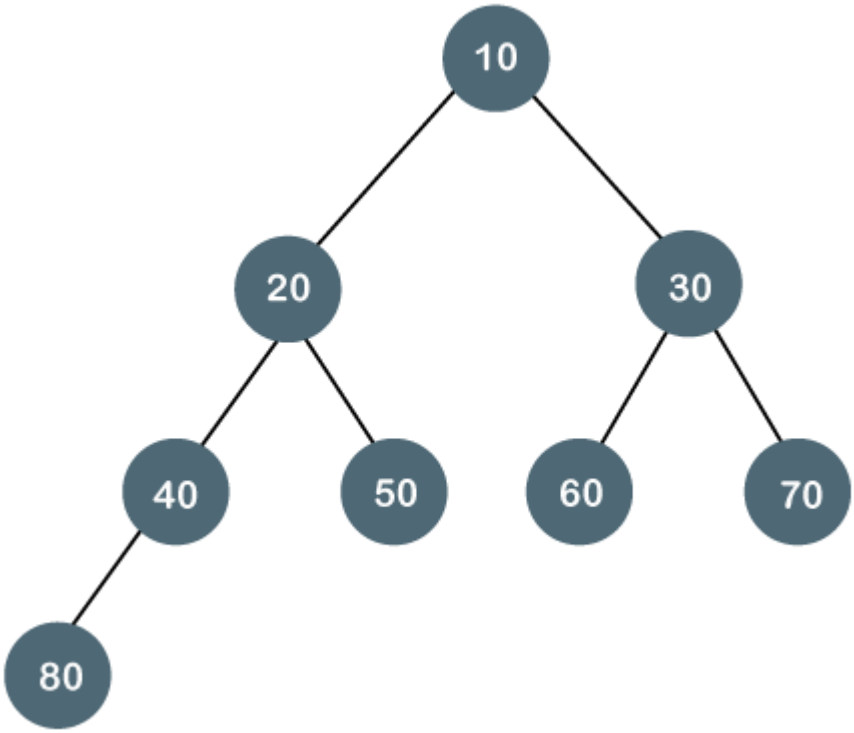
- The maximum height of the full binary tree can be computed as:

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.

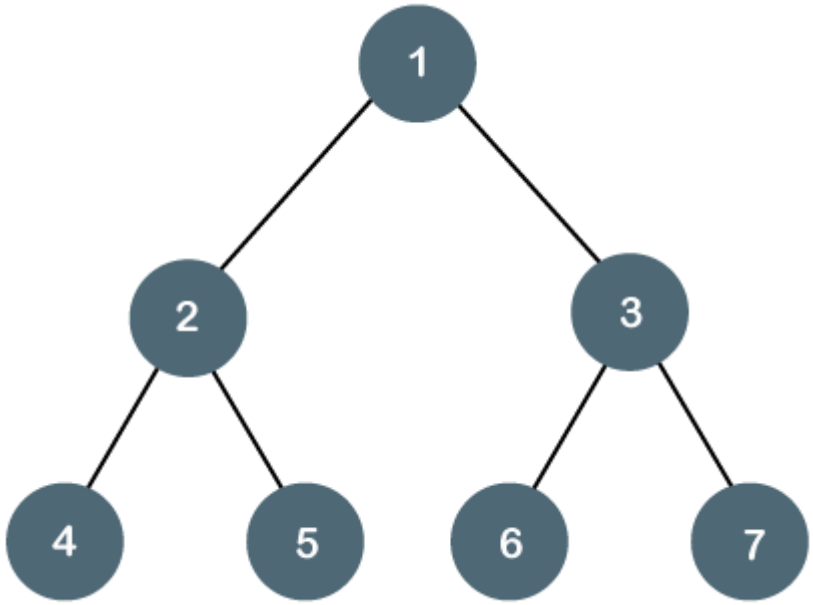
Properties of Complete Binary Tree

- The maximum number of nodes in complete binary tree is 2^{h+1} - 1.

- The minimum number of nodes in complete binary tree is 2^h.
- The minimum height of a complete binary tree is **$\log_2(n+1) - 1$.**
- The maximum height of a complete binary tree is

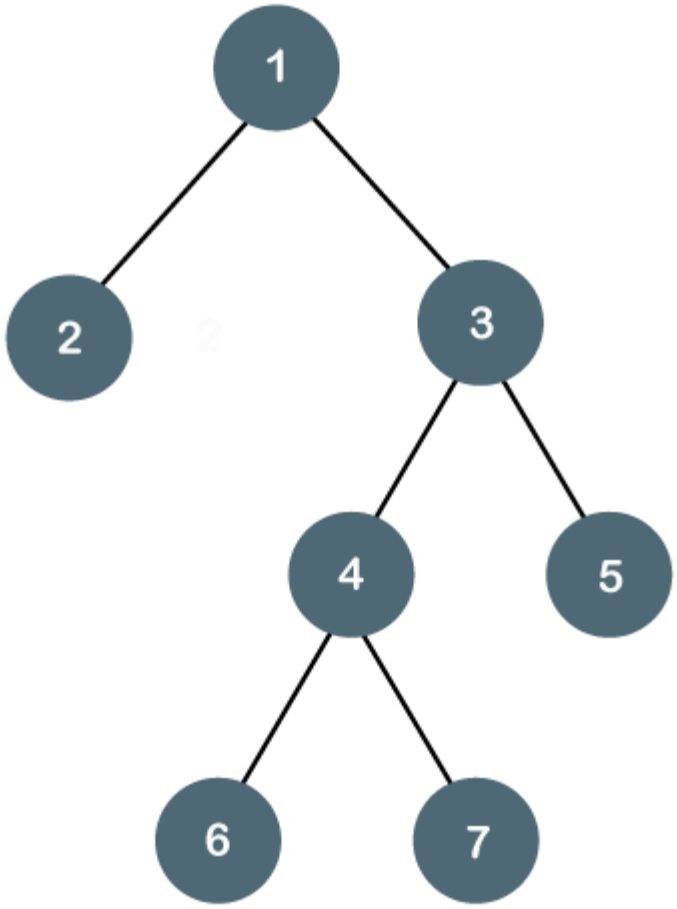
Perfect Binary Tree

A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.



Let's look at a simple example of a perfect binary tree.

The below tree is not a perfect binary tree because all the leaf nodes are not at the same level.

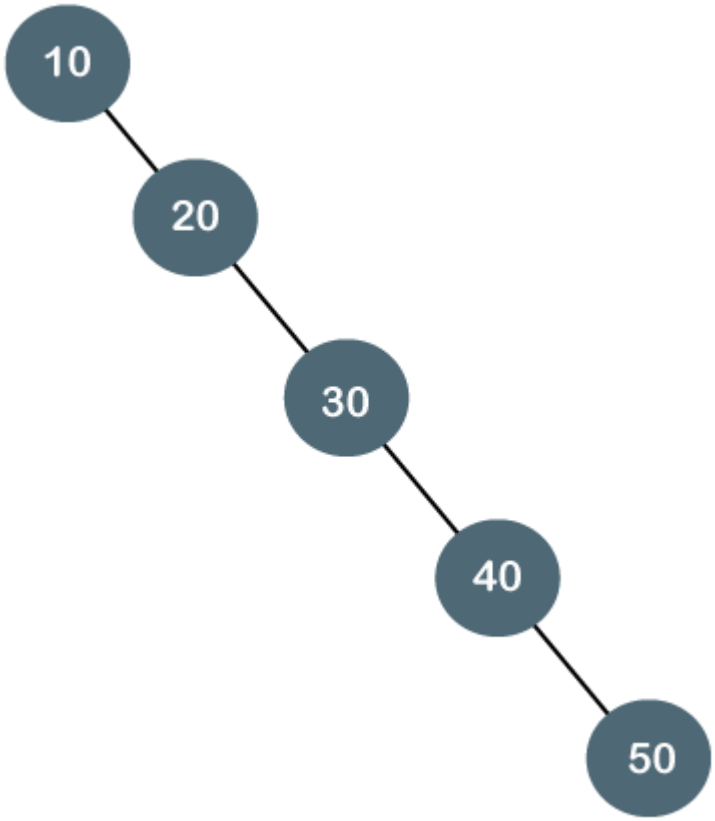


Note: All the perfect binary trees are the complete binary trees as well as the full binary tree, but vice versa is not true, i.e., all complete binary trees and full binary trees are the perfect binary trees.

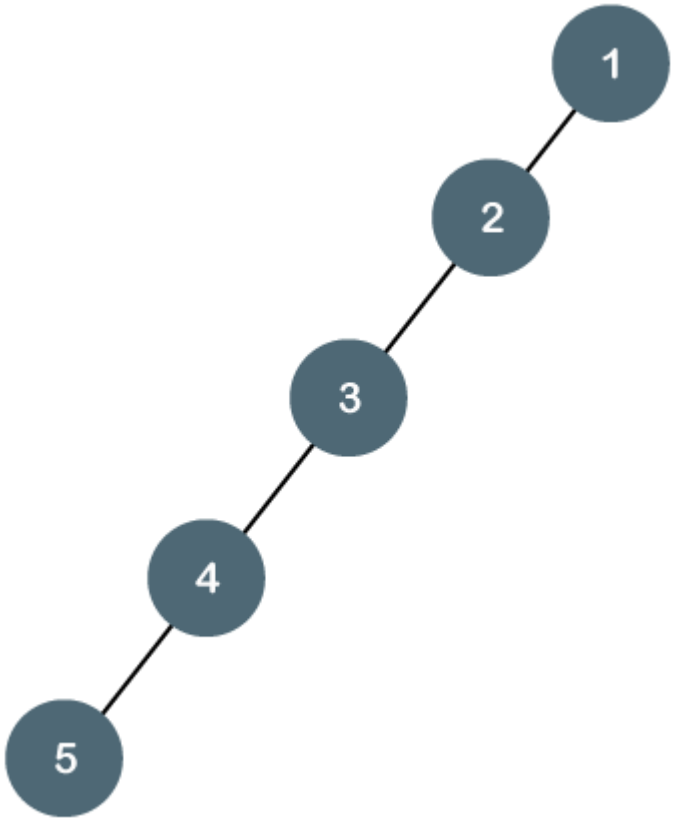
Degenerate Binary Tree

The degenerate binary tree is a tree in which all the internal nodes have only one children.

Let's understand the Degenerate binary tree through examples.



The above tree is a degenerate binary tree because all the nodes have only one child. It is also known as a right-skewed tree as all the nodes have a right child only.

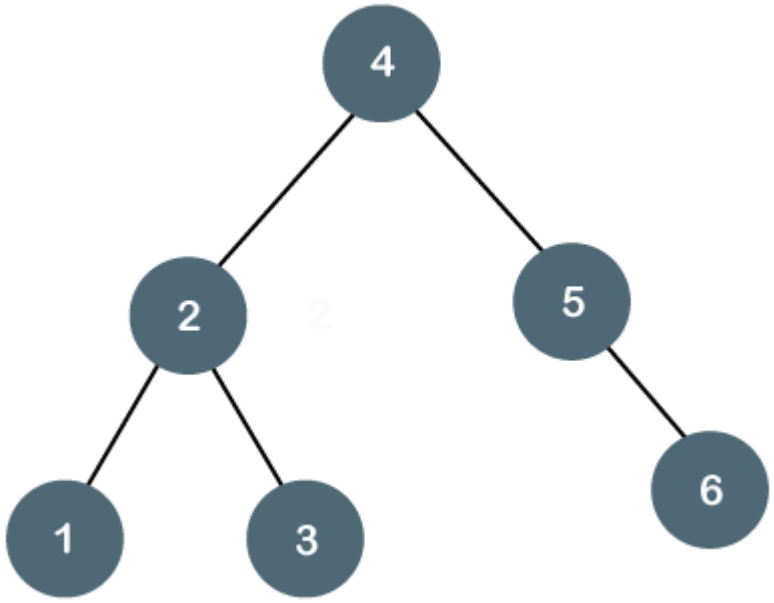


The above tree is also a degenerate binary tree because all the nodes have only one child. It is also known as a left-skewed tree as all the nodes have a left child only.

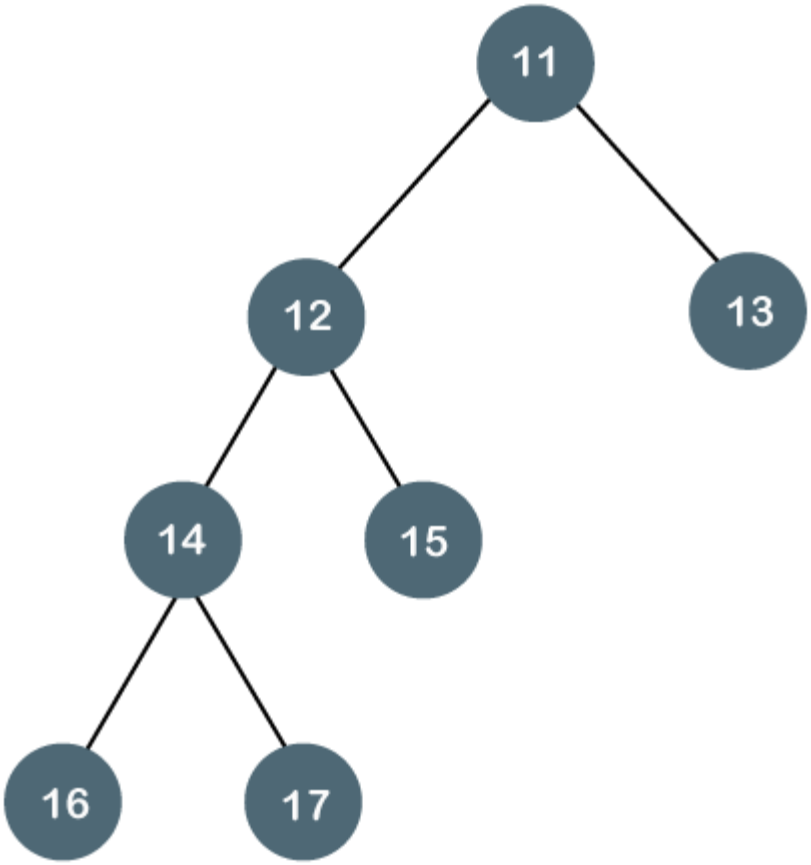
Balanced Binary Tree

The balanced binary tree is a tree in which both the left and right trees differ by atmost 1. For example, ***AVL*** and ***Red-Black trees*** are balanced binary tree.

Let's understand the balanced binary tree through examples.



The above tree is a balanced binary tree because the difference between the left subtree and right subtree is zero.



The above tree is not a balanced binary tree because the difference between the left subtree and the right subtree is greater than 1.

Binary Tree Implementation

A Binary tree is implemented with the help of pointers. The first node in the tree is represented by the root pointer. Each node in the tree consists of three parts, i.e., data, left pointer and right pointer. To create a binary tree, we first need to create the node. We will create the node of user-defined as shown below:

```
1. struct node
2. {
3.    int data,
4.    struct node *left, *right;
5. }
```

In the above structure, **data** is the value, **left pointer** contains the address of the left node, and **right pointer** contains the address of the right node.

Binary Tree program in C

```
1. #include<stdio.h>
2.       struct node
3.      {
4.          int data;
5.          struct node *left, *right;
6.       }
7.       void main()
8.     {
9.        struct node *root;
10.             root = create();
11.          }
12.       struct node *create()
13.       {
```

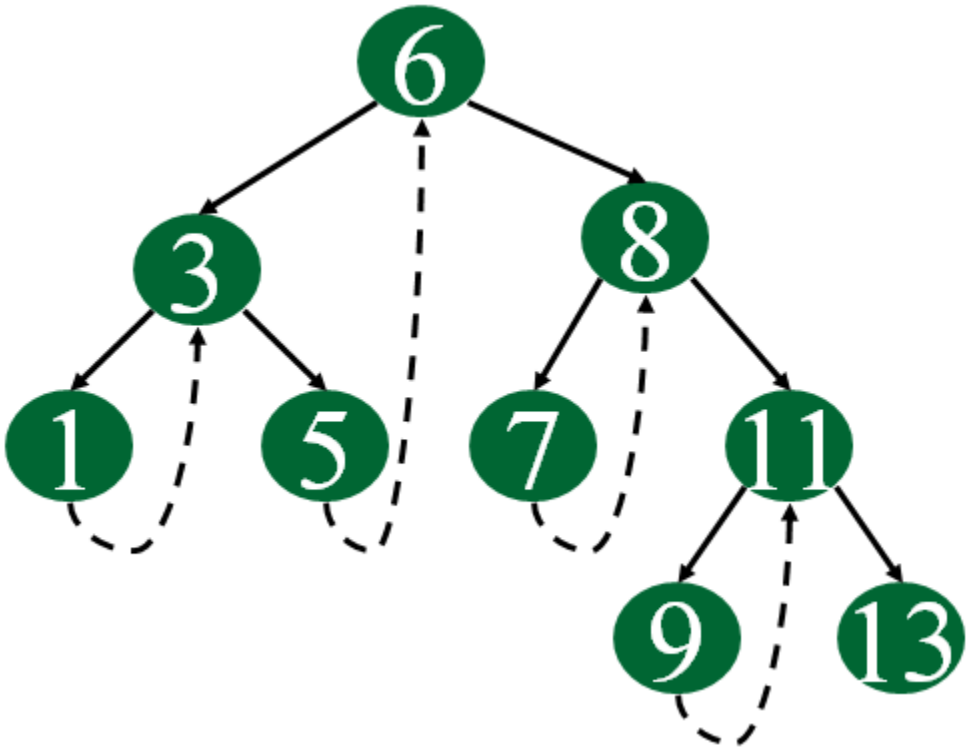
```
14.         struct node *temp;
15.         int data;
16.         temp = (struct node *)malloc(sizeof(struct
    node));
17.         printf("Press 0 to exit");
18.         printf("\nPress 1 for new node");
19.         printf("Enter your choice : ");
20.         scanf("%d", &choice);
21.         if(choice==0)
22.       {
23.       return 0;
24.       }
25.       else
26.       {
27.         printf("Enter the data:");
28.         scanf("%d", &data);
29.         temp->data = data;
30.         printf("Enter the left child of %d", data);
31.         temp->left = create();
32.       printf("Enter the right child of %d", data);
33.       temp->right = create();
34.       return temp;
35.       }
36.       }
```

The above code is calling the create() function recursively and creating new node on each recursive call. When all the nodes are created, then it forms a binary tree structure. The process of visiting the nodes is known as tree traversal. There are three types traversals used to visit a node:

- Inorder traversal
- Preorder traversal
- Postorder traversal

Threaded Binary Tree:

Inorder traversal of a Binary tree can either be done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).
There are two types of threaded binary trees.
Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)
Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
The threads are also useful for fast accessing ancestors of a node.
Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



C representation of a Threaded Node
Following is C representation of a single-threaded node.

- C

```
struct Node
{
    int data;
    struct Node *left, *right;
    bool rightThread;
}
```

Java representation of a Threaded Node
Following is Java representation of a single-threaded node.

- Java
- Python3
- C#
- Javascript

```
static class Node
{
    int data;
    Node left, right;
    boolean rightThread;
}

// This code contributed by aashish1995
```

Since right pointer is used for two purposes, the boolean variable rightThread is used to indicate whether right pointer points to right child or inorder successor. Similarly, we can add leftThread for a double threaded binary tree.

Inorder Traversal using Threads

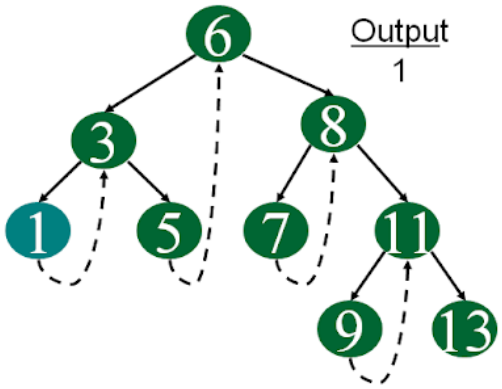
Following is code for inorder traversal in a threaded binary tree.

- C
- Java
- Python3
- C#
- Javascript

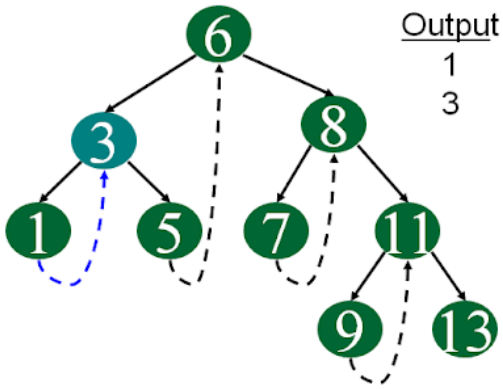
```
// Utility function to find leftmost node in a tree rooted
// with n
struct Node* leftMost(struct Node* n)
{
    if (n == NULL)
        return NULL;

    while (n->left != NULL)
        n = n->left;

    return n;
}

// C code to do inorder traversal in a threaded binary tree
void inOrder(struct Node* root)
{
    struct Node* cur = leftMost(root);
    while (cur != NULL) {
        printf("%d ", cur->data);
```

```
        // If this node is a thread node, then go to
        // inorder successor
        if (cur->rightThread)
            cur = cur->right;
        else // Else go to the leftmost child in right
             // subtree
            cur = leftmost(cur->right);
    }
}
```

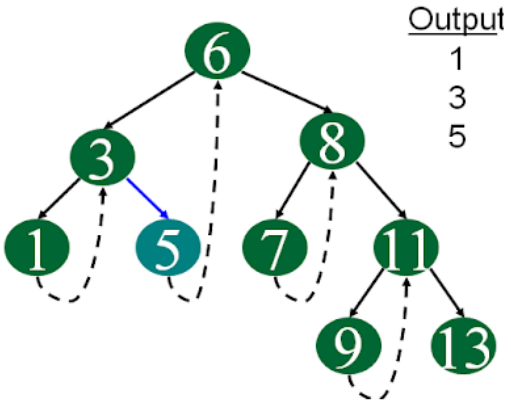
Following diagram demonstrates inorder order traversal using threads.



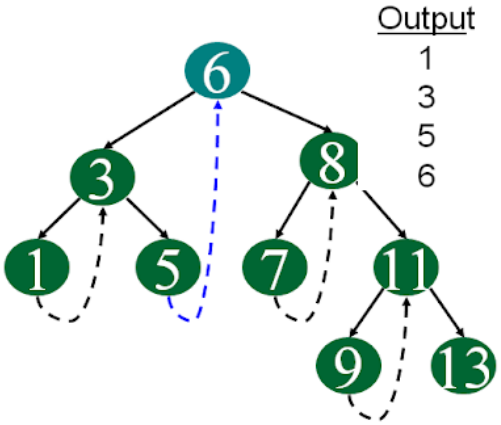
Start at leftmost node, print it



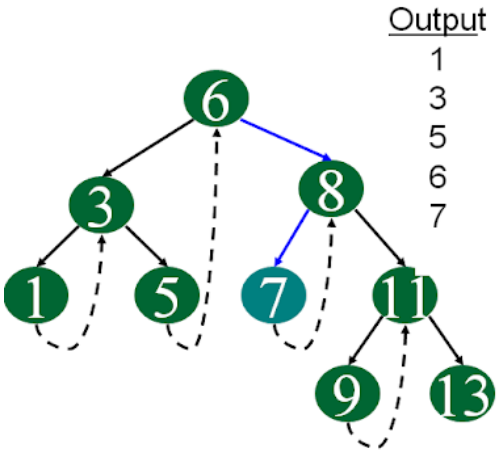
Follow thread to right, print node



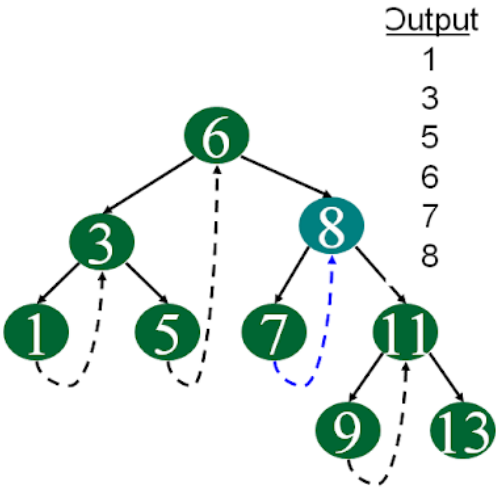
Follow link to right, go to leftmost node and print



Follow thread to right, print node



Follow link to right, go to leftmost node and print



Follow thread to right, print node

continue same way for remaining node.....