Trees and Their Types

Definition of Trees

A **tree** is a hierarchical data structure consisting of nodes, where one node is designated as the root, and the rest are divided into subtrees.

General Tree

A tree where each node can have any number of children.

Binary Tree

A tree where each node has at most two children (left and right).

Basic Terminologies

- **Node**: A single element in a tree.
- **Parent**: A node that has one or more children.
- **Child**: A node directly connected to another node when moving away from the root.
- **Height of a tree**: The length of the longest path from the root to a leaf.
- **Depth of a node**: The number of edges from the root to that node.
- **Leaf Node**: A node without any children.
- **Internal Node**: A node that has at least one child.
- **External Node**: A leaf node (node with no children).

Concepts of Trees

- **Forest**: A collection of disjoint trees.
- **Ordered Tree**: A tree where the relative positions of child nodes matter.
- **Strictly Binary Tree**: A binary tree where each node has either 0 or 2 children.
- **Complete Binary Tree**: A binary tree in which all levels are completely filled except possibly the last level, which is filled from left to right.

Tree Representation

1. **Using Arrays**:

- Suitable for complete binary trees.
- Parent-Child relationships are maintained using index calculations.
- Parent of node at i: arr[(i-1)/2]
- Left child of i: arr[2*i + 1]
- Right child of i: arr[2*i + 2]

2. **Using Linked Lists**:
   - Each node contains data and pointers to its left and right children.
   - More memory efficient for sparse trees.

Binary Tree Traversal Methods

1. Pre-order (NLR)

- Visit **Node** → **Left** subtree → **Right** subtree
- Example: A, B, D, E, C, F

2. In-order (LNR)

- Visit **Left** subtree → **Node** → **Right** subtree
- Example: D, B, E, A, F, C

3. Post-order (LRN)

- Visit **Left** subtree → **Right** subtree → **Node**
- Example: D, E, B, F, C, A

Recursive and Non-Recursive Traversal

- Recursive methods use function calls.
- Non-recursive methods use stacks (for DFS) or queues (for BFS).

Binary Search Tree (BST)

A **BST** is a binary tree where:

- Left subtree nodes < root
- Right subtree nodes > root

Operations on BST

1. **Creation**: Start with an empty tree and add nodes.
2. **Insertion**: Compare values and insert accordingly.
3. **Deletion**:
    - Node with no children: Delete directly.
    - Node with one child: Replace the node with its child.
    - Node with two children: Replace with the in-order successor.

Threaded Binary Trees

- A type of binary tree where unused pointers are used to store references to in-order predecessor and successor.
- Reduces memory usage and eliminates the need for recursion in traversal.

Multi-way Search Trees

- A generalization of BST where a node can have more than two children.
- Example: **B-Trees** used in databases.

Graph Traversal Techniques

1. Breadth-First Search (BFS)

- Explores nodes level by level using a queue.
- Example: Used in shortest path algorithms.

2. Depth-First Search (DFS)

- Explores as deep as possible using a stack (or recursion).
- Example: Used in maze solving.

Heaps

A **Heap** is a complete binary tree used in priority queues.

1. Min Heap

- The parent node is smaller than its children.
- Root contains the minimum element.

2. Max Heap

- The parent node is larger than its children.
- Root contains the maximum element.

- 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.

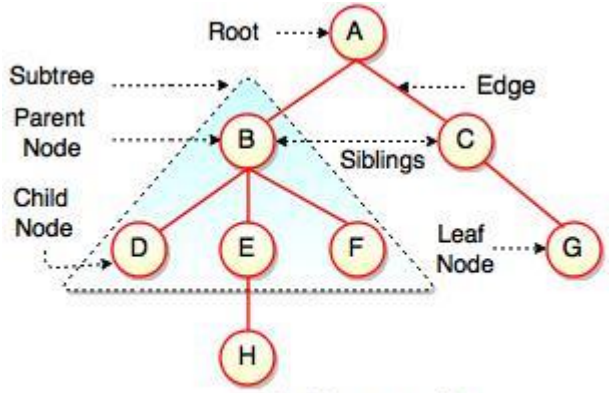

Fig. Structure of Tree

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.

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

In the above figure, D, F, H, G are **leaves**. B and C are **siblings**. Each node excluding a root is connected by a direct edge from exactly one other node
parent → children.

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:

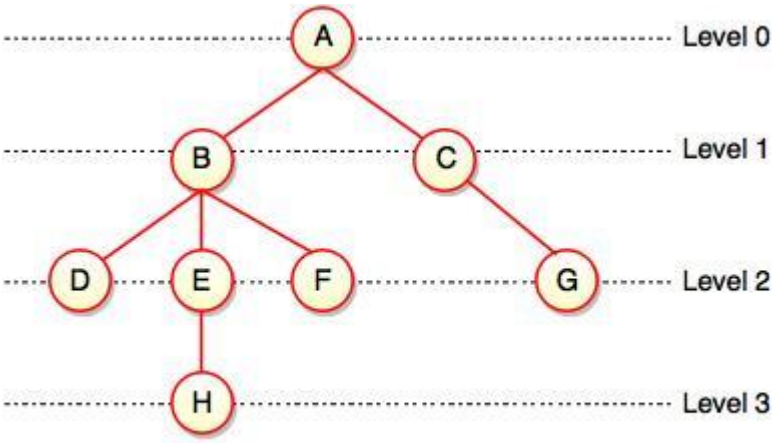

Fig. Levels of Tree

Binary Tree Traversal

Binary tree traversing is a process of accessing every node of the tree and exactly once. A tree is defined in a recursive manner. Binary tree traversal also defined recursively.

There are three techniques of traversal:

1. Preorder Traversal
2. Postorder Traversal
3. Inorder Traversal

1. Preorder Traversal

Algorithm for preorder traversal

Step 1 : Start from the Root.

Step 2 : Then, go to the Left Subtree.

Step 3 : Then, go to the Right Subtree.

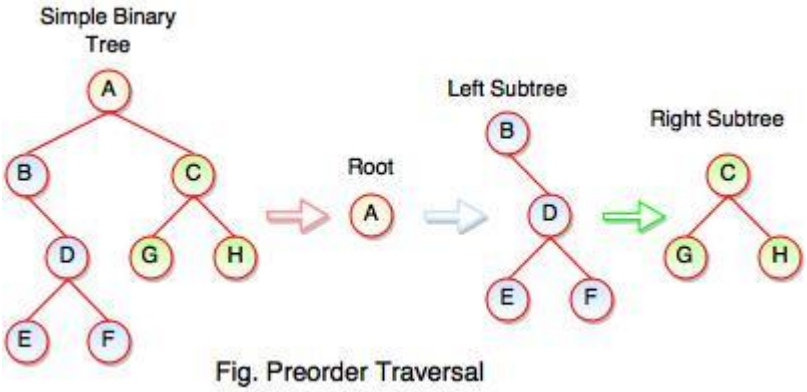

Fig. Preorder Traversal

The above figure represents how preorder traversal actually works.

Following steps can be defined the flow of preorder traversal:

Step 1 : A + B (B + Preorder on D (D + Preorder on E and F)) + C (C + Preorder on G and H)

Step 2 : A + B + D (E + F) + C (G + H)

Step 3 : A + B + D + E + F + C + G + H

Preorder Traversal : A B C D E F G H

2. Postorder Traversal

Algorithm for postorder traversal

Step 1 : Start from the Left Subtree (Last Leaf).

Step 2 : Then, go to the Right Subtree.

Step 3 : Then, go to the Root.

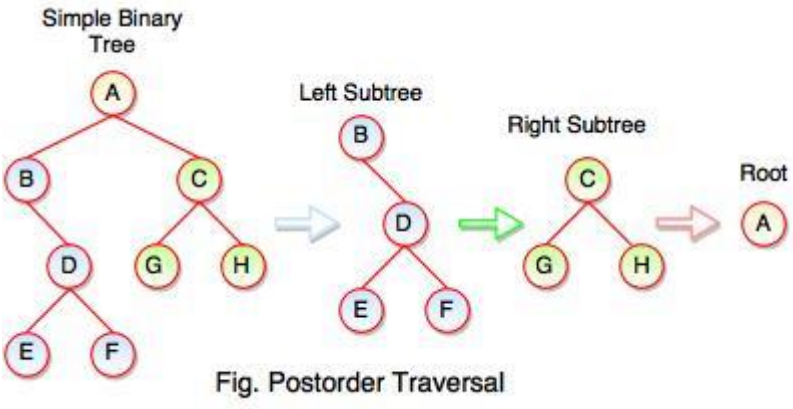

Fig. Postorder Traversal

The above figure represents how postorder traversal actually works.

Following steps can be defined the flow of postorder traversal:

Step 1 : As we know, preorder traversal starts from left subtree (last leaf) ((Postorder on E + Postorder on F) + D + B)) + ((Postorder on G + Postorder on H) + C) + (Root A)

Step 2 : (E + F) + D + B + (G + H) + C + A

Step 3 : E + F + D + B + G + H + C + A

Postorder Traversal : E F D B G H C A

3. Inorder Traversal

Algorithm for inorder traversal

Step 1 : Start from the Left Subtree.

Step 2 : Then, visit the Root.

Step 3 : Then, go to the Right Subtree.

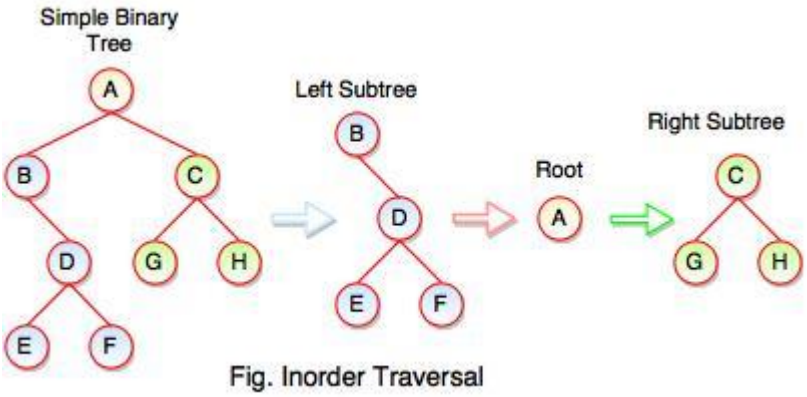

Fig. Inorder Traversal

The above figure represents how inorder traversal actually works.

Following steps can be defined the flow of inorder traversal:

Step 1 : B + (Inorder on E) + D + (Inorder on F) + (Root A) + (Inorder on G) + C (Inorder on H)

Step 2 : B + (E) + D + (F) + A + G + C + H

Step 3 : B + E + D + F + A + G + C + H

Inorder Traversal : B E D F A G C H

Binary Tree

Binary tree is a special type of data structure. In binary tree, every node can have a maximum of 2 children, which are known as **Left child** and **Right Child**. It is a method of placing and locating the records in a database, especially when all the data is known to be in random access memory (RAM).

Definition:

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

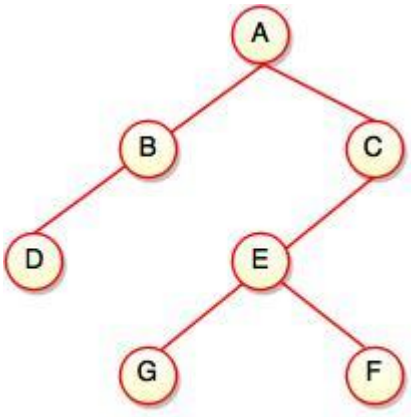

Fig. Binary Tree

The above tree represents binary tree in which node A has two children B and C. Each children have one child namely D and E respectively.

Representation of Binary Tree using Array

Binary tree using array represents a node which is numbered sequentially level by level from left to right. Even empty nodes are numbered.

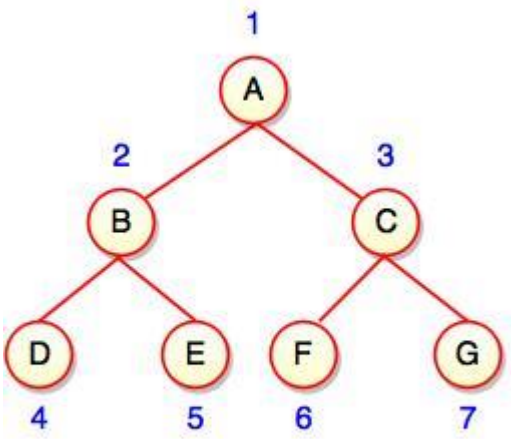

Fig. Binary Tree using Array

Array index is a value in tree nodes and array value gives to the parent node of that particular index or node. Value of the root node index is always -1 as there is no parent for root. When the data item of the tree is sorted in an array, the number appearing against the node will work as indexes of the node in an array.

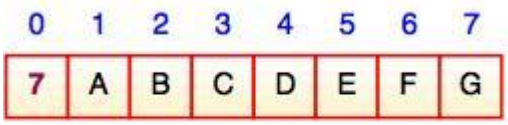

Fig. Location Number of an Array in a Tree

Location number of an array is used to store the size of the tree. The first index of an array that is '0', stores the total number of nodes. All nodes are numbered from left to right level by level from top to bottom. In a tree, each node having an index i is put into the array as its i th element.

The above figure shows how a binary tree is represented as an array. Value '7' is the total number of nodes. If any node does not have any of its child, null value is stored at the corresponding index of the array.

Binary Search Tree

- Binary search tree is a binary tree which has special property called BST.
- BST property is given as follows:

For all nodes A and B,

I. If B belongs to the left subtree of A, the key at B is less than the key at A.

II. If B belongs to the right subtree of A, the key at B is greater than the key at A.

Each node has following attributes:

I. Parent (P), left, right which are pointers to the parent (P), left child and right child respectively.

II. Key defines a key which is stored at the node.

Definition:

"Binary Search Tree is a binary tree where each node contains only smaller values in its left subtree and only larger values in its right subtree."

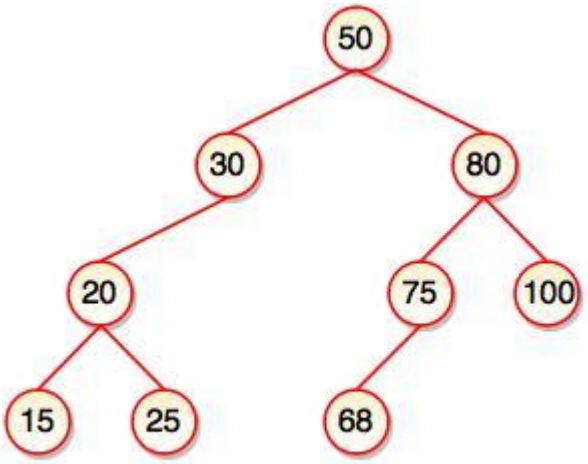

Fig. Binary Search Tree

- The above tree represents binary search tree (BST) where left subtree of every node contains smaller values and right subtree of every node contains larger value.
- Binary Search Tree (BST) is used to enhance the performance of binary tree.
- It focuses on the search operation in binary tree.

Note: Every binary search tree is a binary tree, but all the binary trees need not to be binary search trees.

Binary Search Tree Operations

Following are the operations performed on binary search tree:

1. Insert Operation

- Insert operation is performed with O(log n) time complexity in a binary search tree.
- Insert operation starts from the root node. It is used whenever an element is to be inserted.

The following algorithm shows the insert operation in binary search tree:

Step 1: Create a new node with a value and set its left and right to NULL.

Step 2: Check whether the tree is empty or not.

Step 3: If the tree is empty, set the root to a new node.

Step 4: If the tree is not empty, check whether a value of new node is smaller or larger than the node (here it is a root node).

Step 5: If a new node is smaller than or equal to the node, move to its left child.

Step 6: If a new node is larger than the node, move to its right child.

Step 7: Repeat the process until we reach to a leaf node.

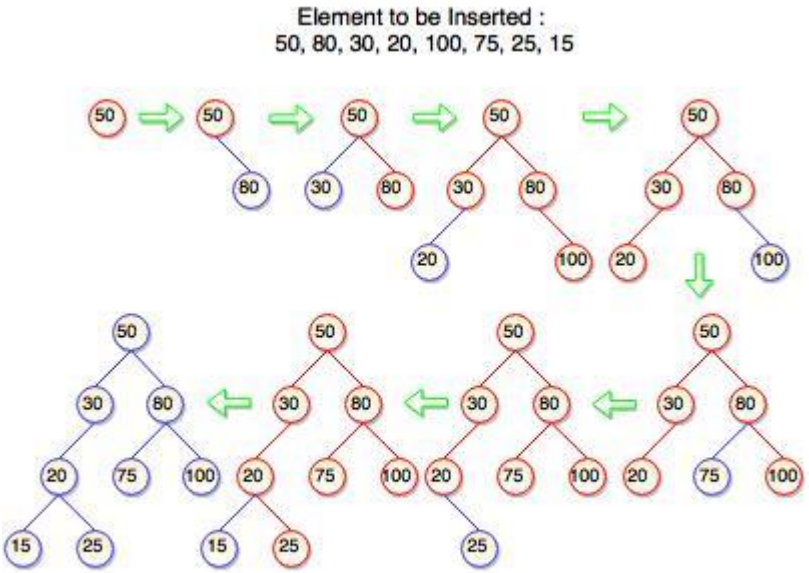

Fig. Insert Operation

The above tree is constructed a binary search tree by inserting the above elements {50, 80, 30, 20, 100, 75, 25, 15}. The diagram represents how the sequence of numbers or elements are inserted into a binary search tree.

2. Search Operation

- Search operation is performed with O(log n) time complexity in a binary search tree.
- This operation starts from the root node. It is used whenever an element is to be searched.

The following algorithm shows the search operation in binary search tree:

Step 1: Read the element from the user .

Step 2: Compare this element with the value of root node in a tree.

Step 3: If element and value are matching, display “Node is Found” and terminate the function.

Step 4: If element and value are not matching, check whether an element is smaller or larger than a node value.

Step 5: If an element is smaller, continue the search operation in left subtree.

Step 6: If an element is larger, continue the search operation in right subtree.

Step 7: Repeat the same process until we found the exact element.

Step 8: If an element with search value is found, display “Element is found” and terminate the function.

Step 9: If we reach to a leaf node and the search value is not match to a leaf node, display “Element is not found” and terminate the function.

There are four types of binary tree:

1. Full Binary Tree
2. Complete Binary Tree
3. Skewed Binary Tree
4. Extended Binary Tree

1. Full Binary Tree

- If each node of binary tree has either two children or no child at all, is said to be a **Full Binary Tree**.
- Full binary tree is also called as **Strictly Binary Tree**.

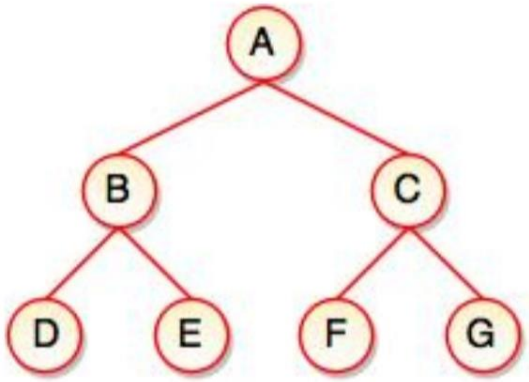

Fig. Full Binary Tree

- Every node in the tree has either 0 or 2 children.
- Full binary tree is used to represent mathematical expressions.

2. Complete Binary Tree

- If all levels of tree are completely filled except the last level and the last level has all keys as left as possible, is said to be a **Complete Binary Tree**.
- Complete binary tree is also called as **Perfect Binary Tree**.

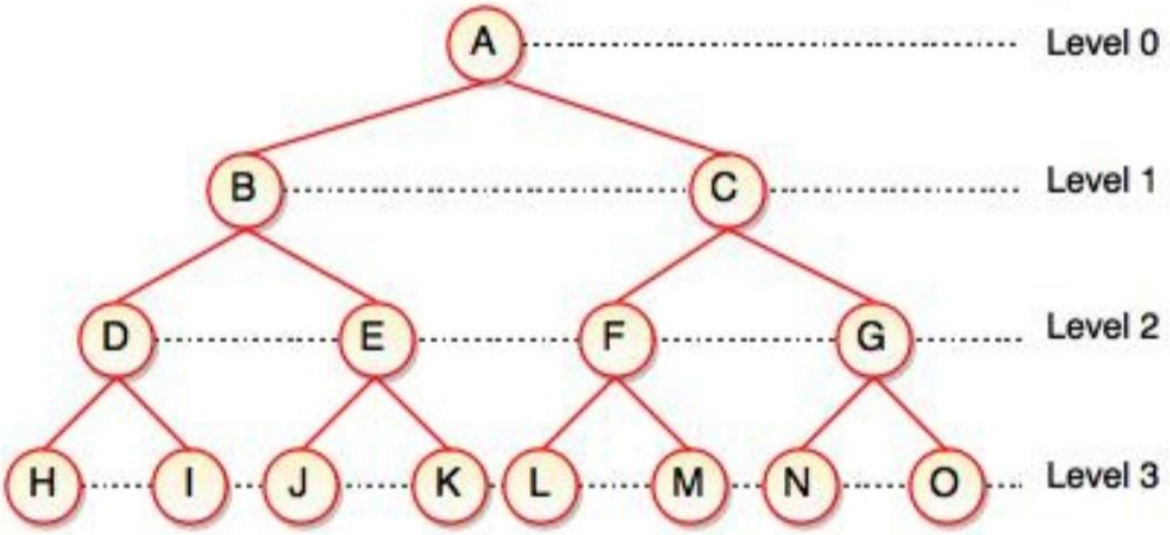

Fig. Complete Binary Tree

- In a complete binary tree, every internal node has exactly two children and all leaf nodes are at same level.
- For example, at Level 2, there must be $2^2 = 4$ nodes and at Level 3 there must be $2^3 = 8$ nodes.

3. Skewed Binary Tree

- If a tree which is dominated by left child node or right child node, is said to be a **Skewed Binary Tree**.
- In a skewed binary tree, all nodes except one have only one child node. The remaining node has no child.

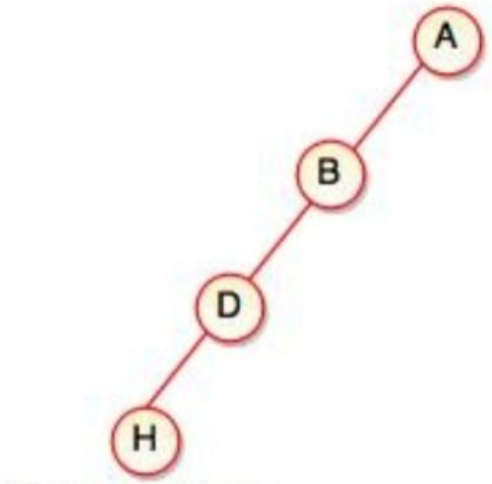

Fig. Left Skewed Binary Tree

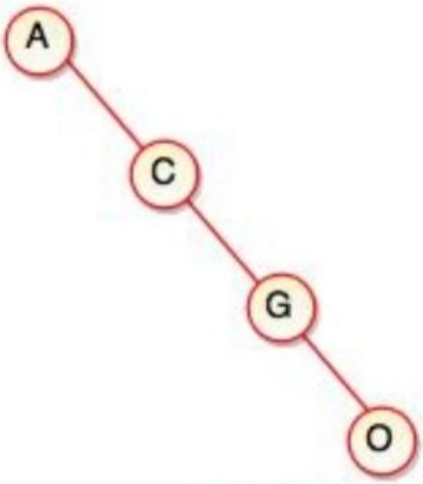

Fig. Right Skewed Binary Tree

- In a left skewed tree, most of the nodes have the left child without corresponding right child.
- In a right skewed tree, most of the nodes have the right child without corresponding left child.

4. Extended Binary Tree

- Extended binary tree consists of replacing every null subtree of the original tree with special nodes.
- Empty circle represents internal node and filled circle represents external node.
- The nodes from the original tree are internal nodes and the special nodes are external nodes.
- Every internal node in the extended binary tree has exactly two children and every external node is a leaf. It displays the result which is a **complete binary tree**.

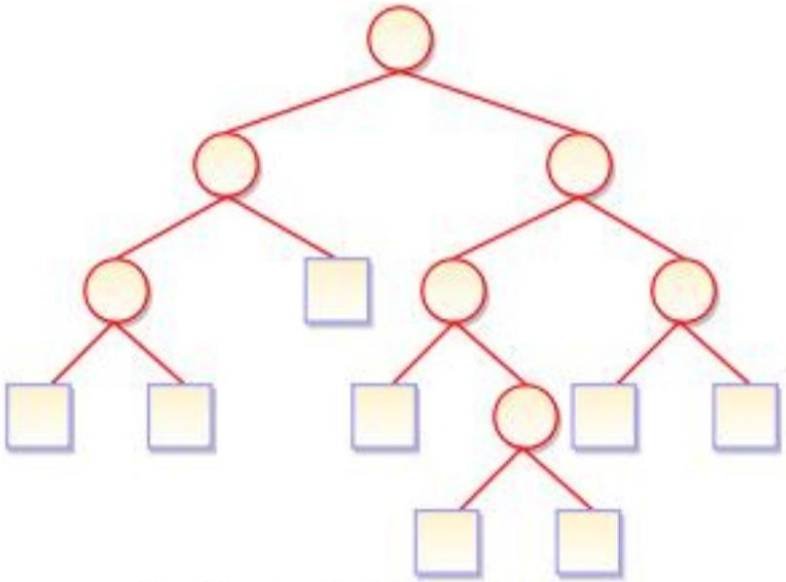

Fig. Extended Binary Tree