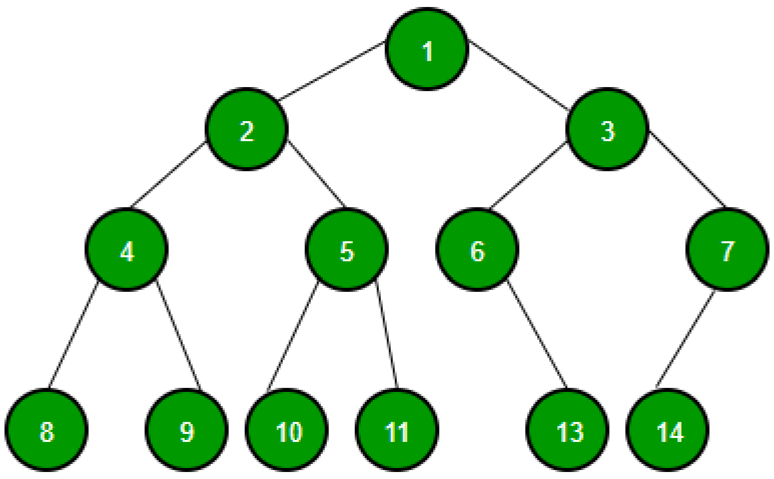
BINARY TREE

A **Binary Tree Data Structure** is a hierarchical data structure in which each node has at most two children, referred to as the left child and the right child. It is commonly used in computer science for efficient storage and retrieval of data, with various operations such as insertion, deletion, and traversal.



Properties of Binary Tree

- **Each node in a binary tree can have at most two child nodes:** In a binary tree, each node can have either zero, one, or two child nodes. If a node has zero children, it is called a leaf node. If a node has one child, it is called a unary node. If a node has two children, it is called a binary node.
- **The node at the top of the tree is called the root node:** The root node is the first node in a binary tree and all other nodes are connected to it. All other nodes in the tree are either child nodes or descendant nodes of the root node.
- **Nodes that do not have any child nodes are called leaf nodes:** Leaf nodes are the endpoints of the tree and have no children. They represent the final result of the tree.

- **The height of a binary tree is defined as the number of edges from the root node to the deepest leaf node:** The height of a binary tree is the length of the longest path from the root node to any of the leaf nodes. The height of a binary tree is also known as its depth.
- **In a full binary tree, every node except the leaves has exactly two children**: In a full binary tree, all non-leaf nodes have exactly two children. This means that there are no unary nodes in a full binary tree.
- **In a complete binary tree, every level of the tree is completely filled except for the last level, which can be partially filled:** In a complete binary tree, all levels of the tree except the last level are completely filled. This means that there are no gaps in the tree and all nodes are connected to their parent nodes.
- **In a balanced binary tree, the height of the left and right subtrees of every node differ by at most 1:** In a balanced binary tree, the height of the left and right subtrees of every node is similar. This ensures that the tree is balanced and that the height of the tree is minimized.
- **The in-order, pre-order, and post-order traversal of a binary tree are three common ways to traverse the tree**: In-order, pre-order, and post-order are three different ways to traverse a binary tree. In-order traversal visits the left subtree, the node itself, and then the right subtree. Pre-order traversal visits the node itself, the left subtree, and then the right subtree. Post-order traversal visits the left subtree, the right subtree, and then the node itself.

1. The maximum number of nodes at level ‘l’ of a binary tree is 2l:
2. The Maximum number of nodes in a binary tree of height ‘h’ is 2h – 1:
3. In a Binary Tree with N nodes, the minimum possible height or the minimum number of levels is Log2(N+1):
4.A Binary Tree with L leaves has at least | Log2L |+ 1 levels:
5. In a Binary tree where every node has 0 or 2 children, the number of leaf nodes is always one more than nodes with two children:
6. In a non-empty binary tree, if n is the total number of nodes and e is the total number of edges, then e = n-1:

Advantages of Binary Tree:

- **Efficient searching**: Binary trees are particularly efficient when searching for a specific element, as each node has at most two child nodes, allowing for binary search algorithms to be used. This means that search operations can be performed in O(log n) time complexity.
- **Ordered traversal:** The structure of binary trees enables them to be traversed in a specific order, such as in-order, pre-order, and post-order. This allows for operations to be performed on the nodes in a specific order, such as printing the nodes in sorted order.
- **Memory efficient**: Compared to other tree structures, binary trees are relatively memory-efficient because they only require two child pointers per node. This means that they can be used to store large amounts of data in memory while still maintaining efficient search operations.
- **Fast insertion and deletion:** Binary trees can be used to perform insertions and deletions in O(log n) time complexity. This makes them a good choice for applications that require dynamic data structures, such as database systems.
- **Easy to implement:** Binary trees are relatively easy to implement and understand, making them a popular choice for a wide range of applications.

- **Useful for sorting:** Binary trees can be used to implement efficient sorting algorithms, such as heap sort and binary search tree sort.

Disadvantages of Binary Tree:

- **Limited structure:** Binary trees are limited to two child nodes per node, which can limit their usefulness in certain applications. For example, if a tree requires more than two child nodes per node, a different tree structure may be more suitable.
- **Unbalanced trees:** Unbalanced binary trees, where one subtree is significantly larger than the other, can lead to inefficient search operations. This can occur if the tree is not properly balanced or if data is inserted in a non-random order.
- **Space inefficiency:** Binary trees can be space inefficient when compared to other data structures. This is because each node requires two child pointers, which can be a significant amount of memory overhead for large trees.
- **Slow performance in worst-case scenarios:** In the worst-case scenario, a binary tree can become degenerate, meaning that each node has only one child. In this case, search operations can degrade to O(n) time complexity, where n is the number of nodes in the tree.
- **Complex balancing algorithms:** To ensure that binary trees remain balanced, various balancing algorithms can be used, such as AVL trees and red-black trees. These algorithms can be complex to implement and require additional overhead, making them less suitable for some applications.

Application of Binary Trees:

- **Search algorithms:** Binary search algorithms use the structure of binary trees to efficiently search for a specific element. The search can be performed in O(log n) time complexity, where n is the number of nodes in the tree.
- **Sorting algorithms:** Binary trees can be used to implement efficient sorting algorithms, such as binary search tree sort and heap sort.

- **Database systems:** Binary trees can be used to store data in a database system, with each node representing a record. This allows for efficient search operations and enables the database system to handle large amounts of data.
- **File systems:** Binary trees can be used to implement file systems, where each node represents a directory or file. This allows for efficient navigation and searching of the file system.
- **Compression algorithms:** Binary trees can be used to implement Huffman coding, a compression algorithm that assigns variable-length codes to characters based on their frequency of occurrence in the input data.
- **Decision trees:** Binary trees can be used to implement decision trees, a type of machine learning algorithm used for classification and regression analysis.
- **Game AI**: Binary trees can be used to implement game AI, where each node represents a possible move in the game. The AI algorithm can search the tree to find the best possible move.

Real-time applications of Binary Trees:

- DOM in HTML.
- File explorer.
- Used as the basic data structure in Microsoft Excel and spreadsheets.
- Editor tool: Microsoft Excel and spreadsheets.
- Evaluate an expression
- Routing Algorithms