TREE DATA STRUCTURES

What is Tree Data Structure?

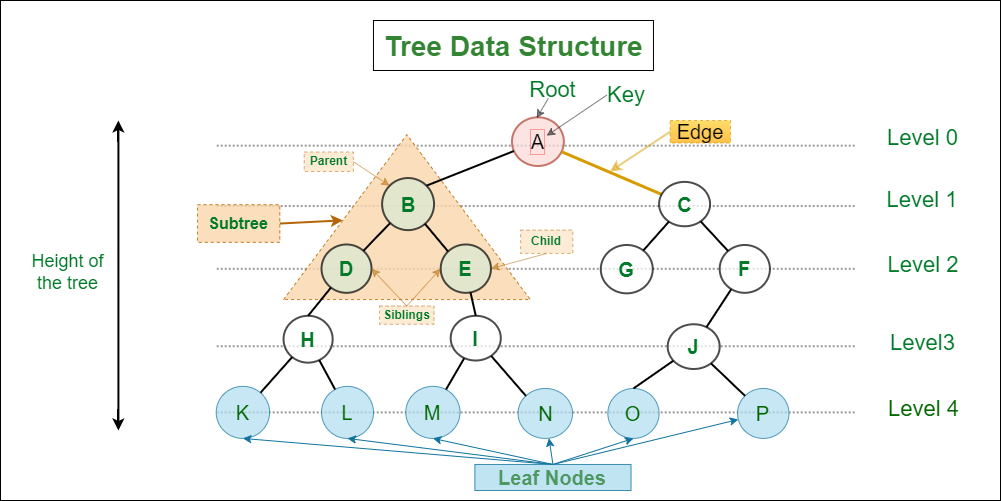
A tree is a type of data structure that represents a hierarchical *relationship between data elements, called nodes. The top node in the tree is called the root, and the elements below the root are called child nodes. Each child node may have one or more child nodes of its own, forming a branching structure. The nodes at the bottom of the tree, which do not have any child nodes, are called leaf nodes.*

*A tree is a non-linear data structure, meaning that elements are not stored in a linear sequence like in an array or a linked list. Instead, elements are organized in a hierarchical structure, with each element having a parent-child relationship with other elements.*

*A tree can be represented in many ways, such as in an array or a linked list, but the most common representation is a graphical one, where each node is represented as a circle and each edge is represented as a line connecting two circles.*

Basic Tree Terminology

* *Node: A fundamental unit in a tree that holds data and may have zero or more child nodes.*
* *Root: The topmost node in a tree, serving as the starting point for traversing the tree.*
* *Parent and Child: Nodes in a tree have relationships where one node is the parent of another, and the connected node is its child.*
* *Leaf: A node with no children, representing the endpoints of a branch in the tree.*
* *Siblings: Nodes that share the same parent are called siblings.*
* *Depth: The level or depth of a node is the length of the path from the root to that node.*
* *Height: The height of a tree is the length of the longest path from the root to a leaf.*



*Tree data structure*

Types of tree data structures.

*There are several tree data structures, each designed for specific purposes and use cases. Here are some common types of tree data structures:*

1. *Binary Tree: Each node has at most two children, typically referred to as the left child and the right child.*
2. *Binary Search Tree (BST): A binary tree in which the left subtree of a node contains only nodes with keys less than the node's key, and the right subtree contains only nodes with keys greater than the node's key.*
3. *AVL Tree: A self-balancing binary search tree in which the height of the two child subtrees of any node differs by at most one.*
4. *Red-Black Tree: Another type of self-balancing binary search tree where each node has an extra bit for denoting the color of the node, either red or black.*
5. *B-Tree: A tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time.*
6. *Trie (Prefix Tree): An ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings.*
7. *Heap: A specialized tree-based data structure that satisfies the heap property. Heaps are commonly used in implementing priority queues.*
8. *Ternary Search Tree: A type of trie where each node has three children, and each edge is labeled with a character.*
9. *Quadtree: A tree data structure in which each internal node has exactly four children, used to represent two-dimensional space.*
10. *Octree: Similar to a quadtree, but in three dimensions. Each internal node has exactly eight children.*
11. *Splay Tree: A self-adjusting binary search tree with the property that recently accessed elements are quick to access again.*
12. *Suffix Tree: A tree-like data structure used for indexing strings to facilitate quick searches for substrings.*

*These are just a few examples, and there are many other variations and specialized tree structures depending on the specific requirements of a problem or application. Each type of tree has its own advantages and use cases based on factors such as efficiency, simplicity, and specific application requirements.*

Why Tree Data Structure?

Unlike Array and Linked List, which are linear data structures, tree is hierarchical (or non-linear) data structure. 

* Hierarchical Structure: Trees are used to model hierarchical structures, such as the file system in a computer or the organizational chart in a company. The tree structure allows for a natural representation of parent-child relationships, making it easy to understand and visualize the data.
* Searching Efficiency: Trees provide an efficient way to search for data. For example, in a binary search tree, searching for a value takes time proportional to the logarithm of the number of elements, which is much faster than searching in a linear data structure like an array or a linked list.
* Sorting: Trees can be used to sort data efficiently. For example, in a self-balancing binary search tree, the data is automatically sorted as it is inserted into the tree, making it easy to find the minimum, maximum, and other values in the tree.
* Dynamic Data: Trees are dynamic data structures, which means that they can grow and shrink as needed. This makes them well-suited for applications where the data changes frequently, such as in real-time systems.
* Efficient Insertion and Deletion: Trees provide efficient algorithms for inserting and deleting data, which is important in many applications where data needs to be added or removed frequently.
* Easy to Implement: Trees are relatively easy to implement, especially when compared to other data structures like graphs. This makes them a popular choice for many programming projects.

Other Applications of Tree Data Structure:

* To Store hierarchical data, like folder structure, organization structure, XML/HTML data.
* [Binary Search Tree](https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/) is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item
* [Heap](https://www.geeksforgeeks.org/heap-data-structure/) is a tree data structure which is implemented using arrays and used to implement priority queues.
* [B-Tree](https://www.geeksforgeeks.org/b-tree-set-1-introduction-2/) and[B+ Tree](https://www.geeksforgeeks.org/database-file-indexing-b-tree-introduction/) : They are used to implement indexing in databases.
* [Syntax Tree](https://www.geeksforgeeks.org/compiler-design-syntax-directed-translation/):  Scanning, parsing , generation of code and evaluation of arithmetic expressions in Compiler design.
* [K-D Tree:](https://www.geeksforgeeks.org/k-dimensional-tree/)A space partitioning tree used to organize points in K dimensional space.
* [Trie](https://www.geeksforgeeks.org/trie-insert-and-search/) : Used to implement dictionaries with prefix lookup.
* [Suffix Tree](https://www.geeksforgeeks.org/pattern-searching-set-8-suffix-tree-introduction/) : For quick pattern searching in a fixed text.
* [Spanning Trees](https://www.geeksforgeeks.org/applications-of-minimum-spanning-tree/) and shortest path trees are used in routers and bridges respectively in computer networks
* As a workflow for compositing digital images for visual effects.
* Decision trees.
* Organization chart of a large organization..
* Machine learning algorithm.
* For indexing in database.
* In Computer Graphics.
* To evaluate an expression.
* In chess game to store defense moves of player.
* In java virtual machine.
* Tree data structures are used to organize and manage files and directories in a file system. Each file and directory is represented as a node in the tree, with parent-child relationships indicating the hierarchical structure of the file system.
* Tree data structures, such as binary search trees, are commonly used to implement efficient searching and sorting algorithms.
* Graphics and UI design
* Tree data structures are commonly used in decision-making algorithms in artificial intelligence, such as game-playing algorithms, expert systems, and decision trees.

**THE TREE TRANSVERSAL ALGORITHM**

**DEFINITION**

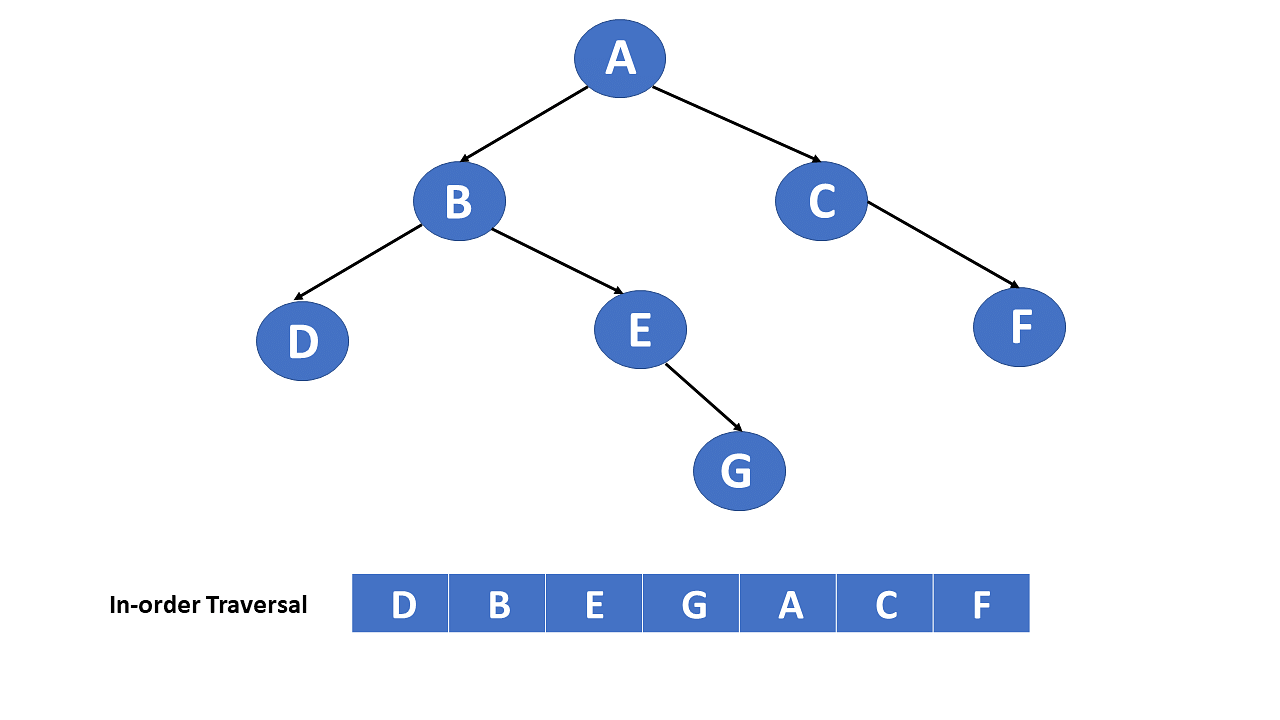
This the process by which each and every element present in a data structure is “visited” (or accessed at least once). We will use an example on the binary tree.

1. **In order transversal**;

-Recursively transvers the left subtree

-visit root node

-Recursively transverse right subtree

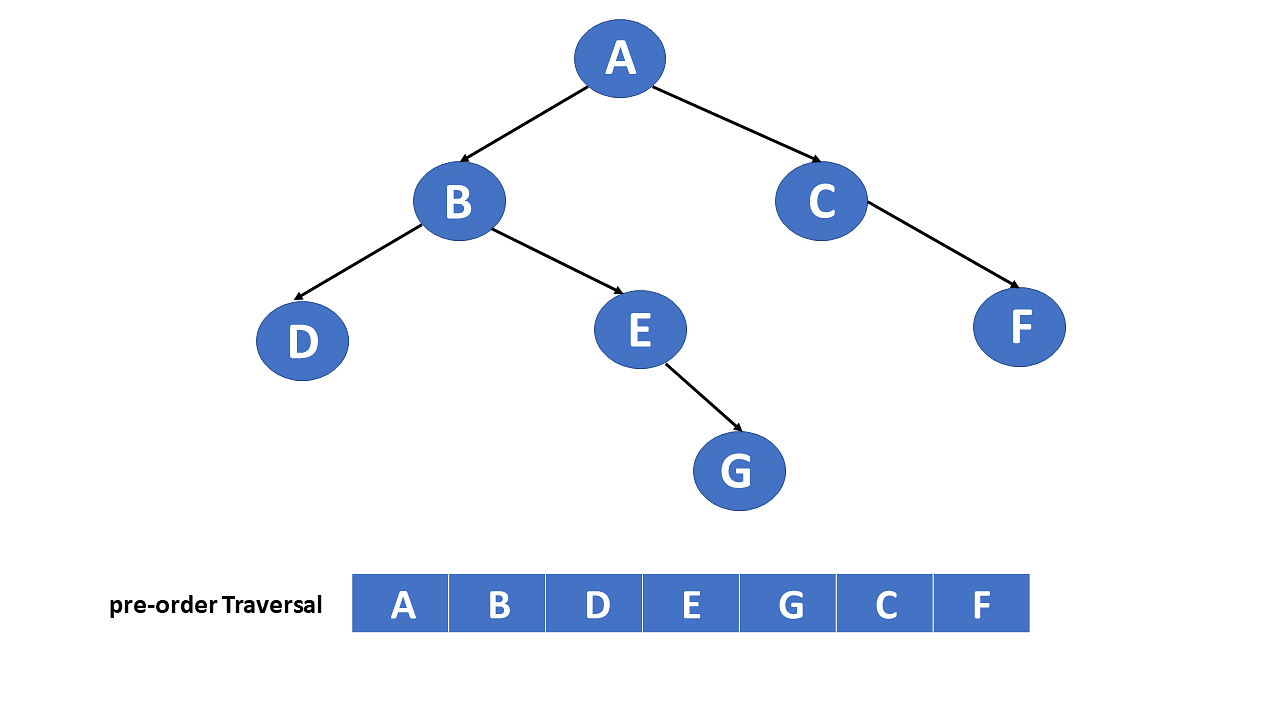


2.**Pre order transversal**;

-Visit root node

-Recursively visit left tree

-Recursively visit right tree

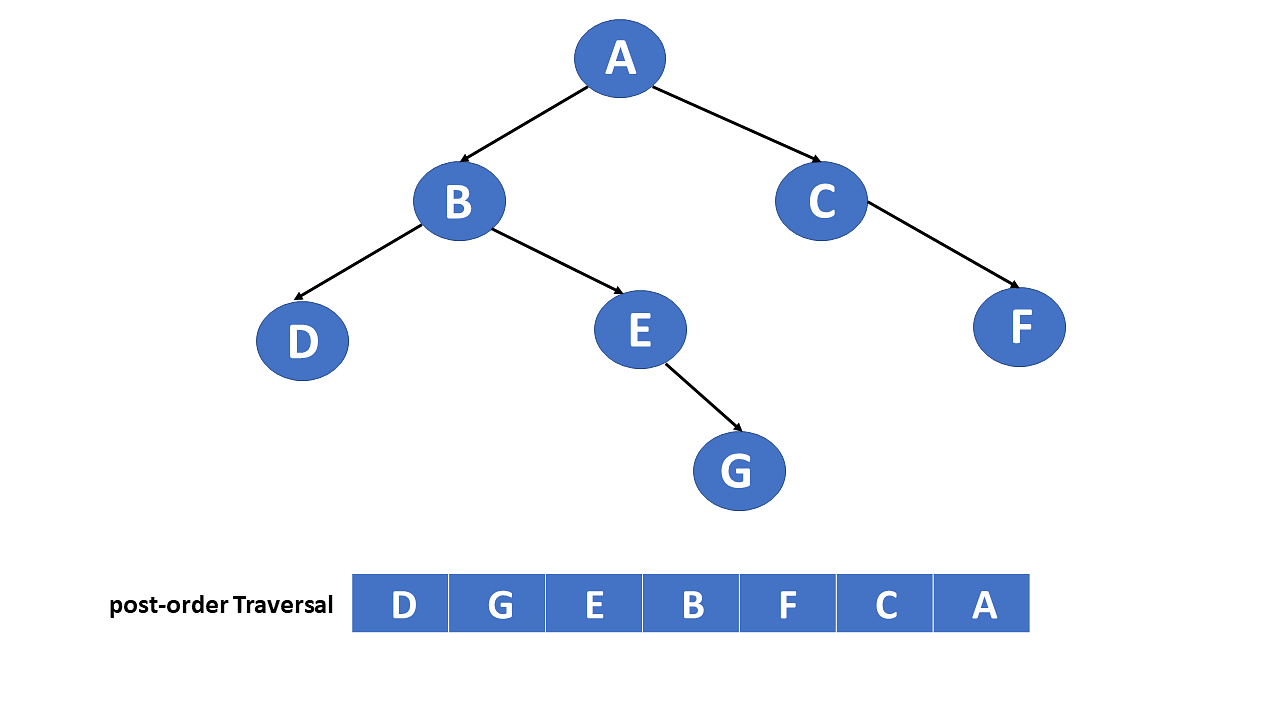


3.**Post order transversal**;

-Transverse left subtree

-Recursively visit transverse right subtree

-Visit root node



**Application of tree transversal algorithms to real life problem**

1.**Network Routing**:

. Tree transversal are use in network routing algorithms to find the most efficient path for data transmission, optimizing the flow of information across networks.

2.**Epidemiology and contact tracing**:

. Representing contact network as tree can help in analyzing and tracing the spread of diseases. Tree transversal assist in identifying individuals or locations critical to the spread.

3.**Supply chain management**:

. Modeling the supply chain as a tree structure allows for efficient management. Tree traversals help in tracking the flow of goods, optimizing logistics, and identifying potential bottlenecks.

4.**Decision Making and Game Theory**:

. Decision trees are used model scenarios in decision-making and game theory. Transversal help in determining optimal choices and outcomes based on different decision paths.

5.**Fraud detection in financial transactions**:

. Representing transaction histories as trees can aid in fraud detection. Transversal helps identify patterns or anomalies in the transaction data.

**ADVANTAGES OF TREE TRANSVERSAL**

1.**Systematic exploration**: There enable systematic exploration of tree structures, ensuring every node is visited exactly once.

2.**Order control**: Transversal algorithm can be adapted to visit nodes in different order, such as pre-order, post-order, in-order, providing flexibility based on specific requirements.

3.**Graph processing**: Tree transversal algorithm are foundational for graph processing, where a tree can be viewed as a special type of graph.

Understanding these advantages helps in choosing the appropriate transversal strategy based on the specific requirements of a given problem or data structure.

**DISADVANTAGES OF TREE TRANSVERSAL**

**1.Not suitable for all operations**: While transversal is effective for task like searching and processing, certain operations, such as finding the height of a tree or determining the lowest common ancestor, may require additional algorithms.

2. **Storage overhead**: Recursive implementations of transverse algorithms can consume a significant amount of stack space, especially for deep or unbalanced trees. Which might lead to a stack overflow.

3. **Complexity**: Understanding and implementing certain transversal algorithms, especially in a non-recursive manner, can be complex and error-prone.

Despite these drawbacks, tree transversal algorithms remain fundamental tools in computer science, and their disadvantages are often mitigated(reduced) through careful consideration of the specific problem an algorithm design.