Trees – Tree Traversals

1. Introduction

Tree Traversal is the process of **visiting every node of a tree exactly once** in a specific order.

Since trees are **non-linear data structures**, traversal is required to access and process all nodes.

Different traversal techniques are used depending on the **problem requirement**.

---

2. What is Tree Traversal?

Tree traversal means:

- Starting from the **root node**
- Visiting each node systematically
- Following a defined order

Traversal helps in:

- Printing tree elements
- Searching nodes
- Evaluating expressions
- Performing tree-based algorithms

---

3. Types of Tree Traversals

Tree traversals are mainly classified into:

1.

Depth-First Traversal (DFS)

- Preorder

- Inorder
- Postorder

2.

Breadth-First Traversal (BFS)

- Level Order

---

4. Preorder Traversal (Root → Left → Right)

Order

```
Root → Left Subtree → Right Subtree
```

Plain English Logic

1. Visit the root node
2. Traverse the left subtree
3. Traverse the right subtree

Example Tree

```
  A
 / \
B   C
```

Preorder Output:

```
A B C
```

---

5. Inorder Traversal (Left → Root → Right)

Order

```
Left Subtree → Root → Right Subtree
```

Plain English Logic

1. Traverse the left subtree
2. Visit the root node
3. Traverse the right subtree

Example Output

```
B A C
```

> Inorder traversal of a
>
> **Binary Search Tree**
>
> **sorted output**

---

6. Postorder Traversal (Left → Right → Root)

Order

```
Left Subtree → Right Subtree → Root
```

Plain English Logic

1. Traverse the left subtree
2. Traverse the right subtree
3. Visit the root node

Example Output

```
B C A
```

---

7. Level Order Traversal (Breadth-First)

Order

> Level by level from left to right

Plain English Logic

1. Visit nodes level-wise
2. Use a queue to store child nodes
3. Process nodes in FIFO order

Example Output

> A B C

---

8. Visualization Summary

Traversal	Order
Preorder	Root → Left → Right
Inorder	Left → Root → Right
Postorder	Left → Right → Root
Level Order	Level by level

---

9. Why Multiple Traversals?

Different traversals are useful for different tasks:

Traversal	Use Case
Preorder	Copy tree, prefix expressions
Inorder	Sorted output (BST)
Postorder	Deleting tree, postfix expressions

Traversal	Use Case
Level Order	Shortest path, BFS

10. Time and Space Complexity

Traversal	Time Complexity	Space Complexity
DFS (All)	O(n)	O(h)
BFS	O(n)	O(n)

Where:

- n = number of nodes
- h = height of tree

11. Recursive Nature of Traversals

Tree traversals are naturally **recursive** because:

- Each subtree is itself a tree
- Traversal logic repeats at each node

12. Applications of Tree Traversals

- Expression evaluation
- File system navigation
- Syntax tree processing
- Searching and sorting
- Artificial Intelligence
- Compilers

13. Advantages of Tree Traversals

- Systematic node access

- Flexible traversal orders
- Efficient tree processing
- Works for all tree types

---

14. Limitations

- Recursive traversal may cause stack overflow
- Level order needs extra memory
- Complex for beginners initially

---

15. Summary

- Traversal means visiting all nodes
- DFS and BFS are main categories
- Preorder, Inorder, Postorder, Level Order are commonly used
- Time complexity is O(n)
- Essential for tree algorithms
