CBS Department

DFS and BFS Traversal Algorithms

Soma Soleiman Zadeh

Data Structures & Algorithms (CBS 216)

Spring 2025 - 2026

Week 15

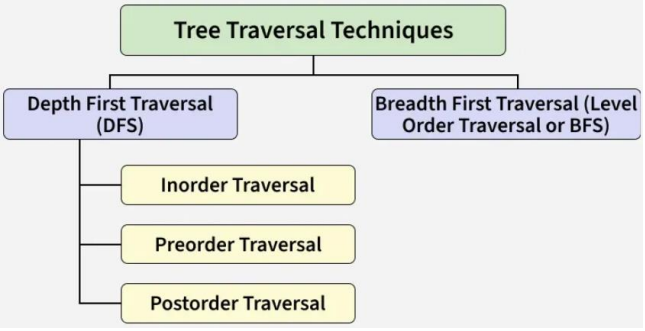
May 14, 2026

Outline

- Tree Traversal Techniques
- **Depth-First Search (DFS)**
  - In-Order Traversal
  - Pre-Order Traversal
  - Post-Order Traversal
- **Breadth-First Search (BFS)**

What is Tree Traversal?

- **Tree traversal** is the process of visiting or accessing each node of a tree exactly once in a specific order.
- Trees offer multiple ways to traverse their nodes.
  - **Depth-First Search (DFS)**
  - **Breadth-First Search (BFS)**

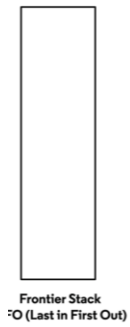


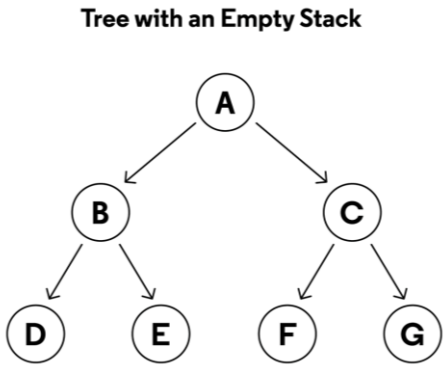
Depth-First Search (DFS)

- **Depth-First Search (DFS)** is an algorithm used to traverse or search through a data structure like a tree or a graph.
- It starts at the **root** node and tries to go down as far as possible until reaching a **leaf node**.
- When it reaches a leaf node, it **backtracks** to the parent node to explore the next path.

Depth-First Search (DFS)

- There are two main ways to implement a **DFS**:
  1. Using the **Recursion** approach
  2. Using **Stack Data Structure**





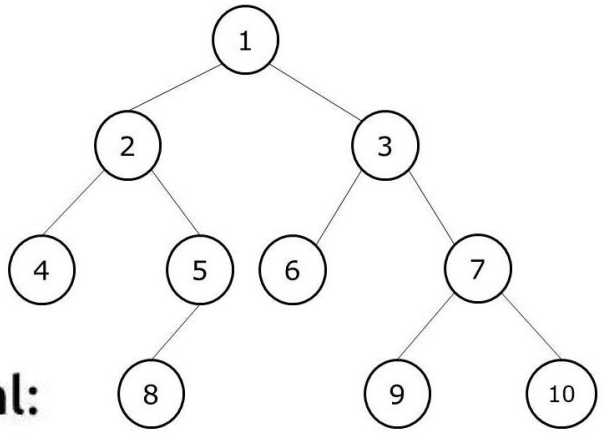
When to Use Depth-First Search (DFS)?

- **DFS** is useful in problems where you need to **explore every possible solution**.
- Some Applications of **DFS**:
  - **Pathfinding problems**, like navigating a game board or finding routes on a map, require a lot of searches.
  - **Navigating decision trees in AI**, where each branch represents a sequence of choices, and the goal is to evaluate all possible outcomes.
- **DFS** ensures that every node is visited once and that the algorithm covers the entire tree.

DFS Techniques

- There are three ways to traverse a tree using **DFS**:
  - **In-order** traversal
  - **Pre-order** traversal
  - **Post-order** traversal
- Each of these implementations are **DFS** and explore down a path fully. **The only difference is the order in which they use the current node's data**.

In-Order Traversal



Inorder Traversal:

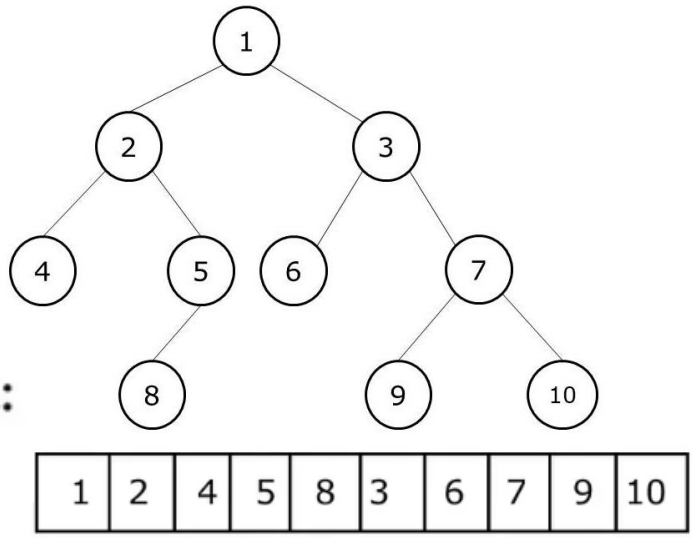
[left,root,right]

4	2	8	5	1	6	3	9	7	10

In-Order Traversal Implementation

```python
def inorder(root):
    if root is None:
        return []

    result = []
    result.extend(inorder(root.left))
    result.append(root.value)
    result.extend(inorder(root.right))

    return result
```

Pre-Order Traversal



Preorder Traversal:

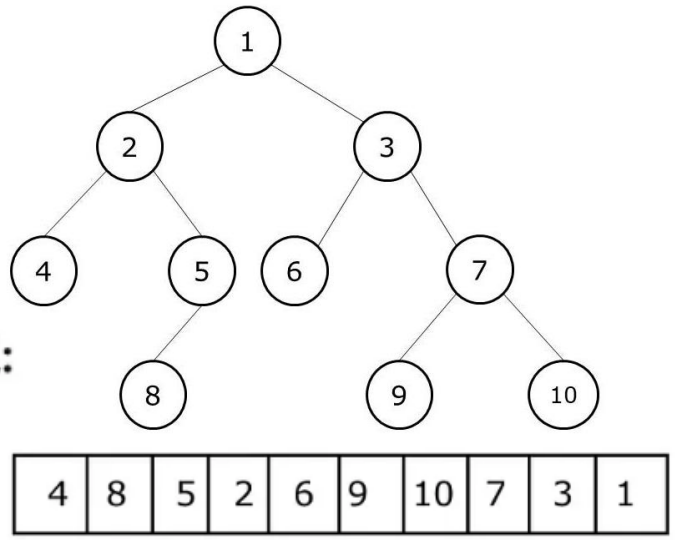
[root, left, right]

1	2	4	5	8	3	6	7	9	10

Pre-Order Traversal Implementation

```
def preorder(root):
    if root is None:
        return []

    result = []
    result.append(root.value)
    result.extend(preorder(root.left))
    result.extend(preorder(root.right))

    return result
```

Post-Order Traversal



Postorder Traversal:

[left, right, root]

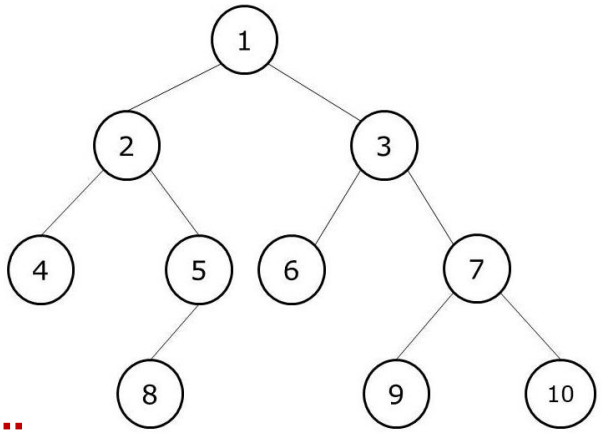
Post-Order Traversal Implementation

```
def postorder(root):
    if root is None:
        return []

    result = []
    result.extend(postorder(root.left))
    result.extend(postorder(root.right))
    result.append(root.value)

    return result
```

Breadth-First Search (BFS)

- **Breadth-First Search (BFS)** is a traversal algorithm that explores a tree or graph level by level.
- It starts from the **root** node, then visits all its immediate neighbors (at distance 1) before moving on to the next level of nodes.
- This ensures that nodes at the same depth are processed before going deeper. Because of this, **BFS** is useful for problems that require examining all possible paths in a structured, systematic way.

BFS Traversal



BFS Traversal:

Root, level 1, level 2, ...

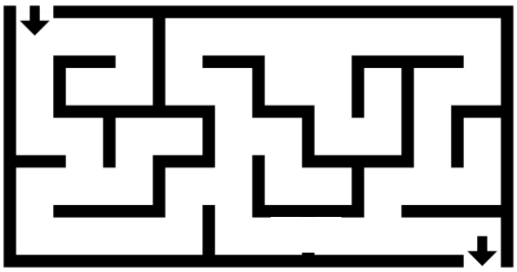
1	2	3	4	5	6	7	8	9	10

BFS Traversal Implementation

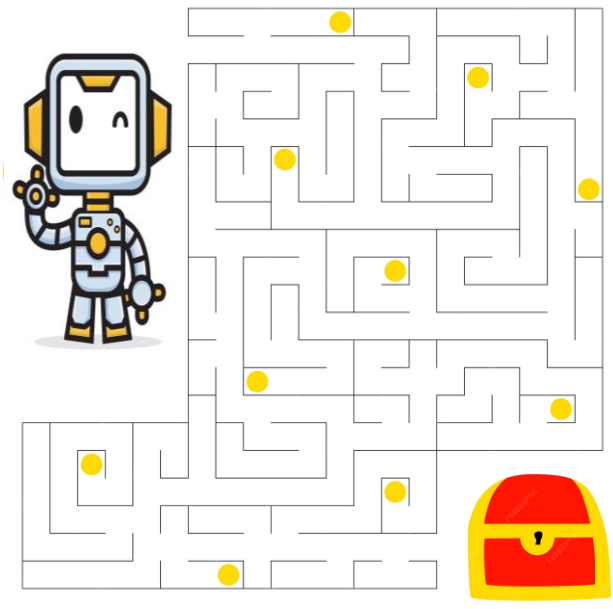
```
from collections import deque
def BFS(root):
    if root is None:
        return []

    result = []
    queue = deque([root])

    while queue:
        level_size = len(queue)
        visited = []

        for i in range(level_size):
            node = queue.popleft()
            visited.append(node.value)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        result.append(visited)
    return result
```

When to Use Breadth-First Search (BFS)?

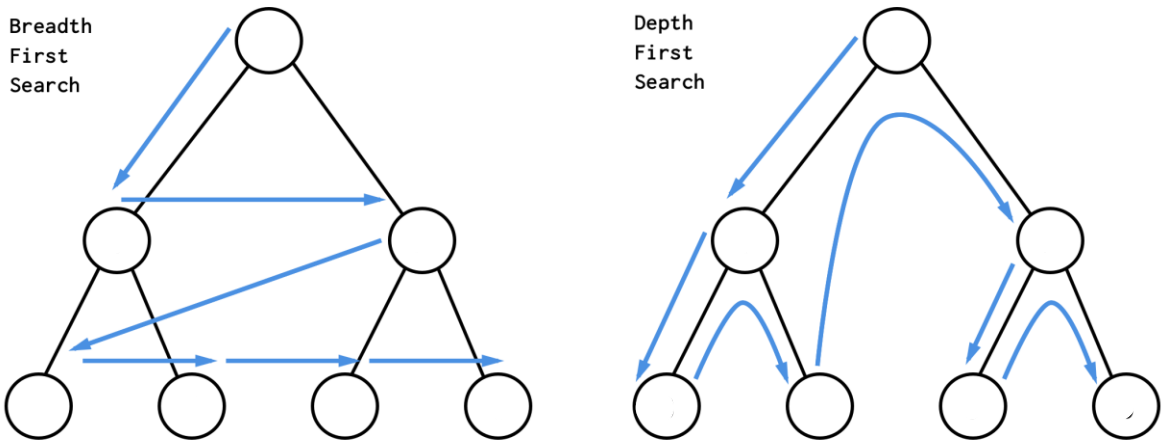
- **Finding the Shortest Path Between Nodes:**
  - **Shortest Path in a Maze**
  - **Network Routing**, where the goal is to find the most efficient route between two points.



A Robotics Problem



DFS vs. BFS



Time Complexity

Traversal	Time Complexity	Use Case
DFS	$O(n)$	Exploring deep branches or pathfinding to leaves.
BFS	$O(n)$	Finding the shortest path or nodes closest to the root

Class Work

- **DFS Traversal**
  - **In-Order** Traversal?
  - **Pre-Order** Traversal?
  - **Post-Order** Traversal?
- **BFS Traversal?**