**Breadth-First Search (BFS)**

**Definition:**

BFS is an algorithm for traversing or searching graph data structures. It starts at a given node (often referred to as the root) and explores all its neighboring nodes at the current depth before moving to nodes at the next depth level.

**Key Characteristics:**

1. **Level-wise Traversal:** BFS traverses all nodes at one depth level before proceeding to the next level.
2. **Queue Data Structure:** It uses a queue to track nodes yet to be explored.
3. **Time Complexity:** O(V+E)O(V + E), where VV is the number of vertices and EE is the number of edges.
4. **Applications:**
   * Finding the shortest path in unweighted graphs.
   * Crawling web pages in search engines.
   * Broadcasting in networks.

**Steps:**

1. Mark the starting node as visited and enqueue it.
2. While the queue is not empty:
   * Dequeue a node and explore its neighbors.
   * For each unvisited neighbor, mark it as visited and enqueue it.

**Depth-First Search (DFS)**

**Definition:**

DFS is an algorithm for traversing or searching graph data structures. It starts at a source node and explores as far as possible along each branch before backtracking.

**Key Characteristics:**

1. **Recursive or Iterative Approach:** DFS can be implemented using recursion or a stack.
2. **Stack Data Structure:** A stack (explicit or implicit through recursion) is used for backtracking.
3. **Time Complexity:** O(V+E)O(V + E), where VV is the number of vertices and EE is the number of edges.
4. **Applications:**
   * Detecting cycles in graphs.
   * Solving puzzles and mazes.
   * Topological sorting.

**Steps:**

1. Mark the starting node as visited.
2. Recursively explore each unvisited neighbor.
3. Backtrack to the previous node when all neighbors are explored.

**Comparison Between BFS and DFS**

| **Feature** | **BFS** | **DFS** |
| --- | --- | --- |
| **Data Structure Used** | Queue | Stack (explicit or implicit via recursion) |
| **Traversal** | Level-wise | Depth-wise |
| **Best Use Case** | Shortest path or minimal edge traversal | Exploring entire paths |
| **Memory Requirement** | More (stores all neighbors in queue) | Less (stores one path in stack) |