

# Graphs – Depth First Search (DFS)

## 1. Introduction

Depth First Search (DFS) is a **graph traversal algorithm** that explores a graph by going **as deep as possible** along each path before backtracking.

Unlike BFS, which explores level by level, DFS explores **one branch completely** before moving to another.

DFS is widely used in **cycle detection, path finding, and topological sorting**.

---

## 2. What is DFS?

DFS starts from a **source vertex** and:

- Visits the vertex
- Moves to one unvisited neighbor
- Continues deeper until no unvisited neighbors remain
- Backtracks to explore other paths

DFS uses:

- **Recursion** or
  - **Explicit stack**
- 

## 3. Key Characteristics of DFS

- Traverses graph **depth-wise**
- Uses **stack (LIFO)** or recursion
- Goes deep before exploring siblings
- Does not guarantee shortest path
- Suitable for deep graph exploration

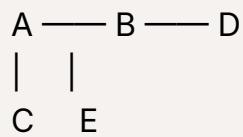
---

## 4. DFS Working Principle (Concept)

1. Start from a source node
  2. Mark it as visited
  3. Visit an unvisited adjacent node
  4. Repeat until no unvisited neighbors
  5. Backtrack and continue
- 

## 5. DFS Example

Graph:



Start DFS from **A**

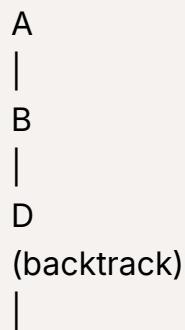
Traversal order (one possible order):

```
A → B → D → E → C
```

DFS explores one branch fully before switching.

---

## 6. Visualization of DFS Traversal



```
E  
(backtrack)  
|  
C
```

## 7. DFS Using Stack (Conceptual)

Stack state:

```
[A]  
[A, B]  
[A, B, D]  
(backtrack)  
[A, B, E]  
(backtrack)  
[A, C]
```

## 8. DFS Algorithm (Plain English)

1. Mark the starting vertex as visited
2. Visit it
3. For each adjacent vertex:
  - If unvisited, apply DFS recursively
4. Continue until all reachable vertices are visited

## 9. Time and Space Complexity

Aspect	Complexity
Time Complexity	$O(V + E)$
Space Complexity	$O(V)$

Where:

- $V$  = number of vertices
  - $E$  = number of edges
- 

## 10. DFS vs BFS

Feature	DFS	BFS
Data Structure	Stack / Recursion	Queue
Traversal	Depth-wise	Level-wise
Shortest Path	No	Yes (unweighted)
Memory Usage	Less	More

---

## 11. Applications of DFS

- Cycle detection
  - Topological sorting
  - Path existence checking
  - Maze solving
  - Connected components
  - Graph coloring
- 

## 12. Advantages of DFS

- Simple implementation
  - Less memory than BFS
  - Useful for deep graphs
  - Effective for connectivity problems
- 

## 13. Limitations of DFS

- Does not guarantee shortest path

- Can go too deep (stack overflow)
  - Order of traversal varies
  - Harder to visualize than BFS
- 

## 14. DFS in Trees vs Graphs

- In trees, DFS corresponds to **preorder, inorder, and postorder traversals**
  - In graphs, a visited array is required to avoid infinite loops
- 

## 15. Summary

- DFS explores graph deeply before backtracking
  - Uses recursion or stack
  - Time complexity is  $O(V + E)$
  - Suitable for depth-oriented problems
  - One of the most fundamental graph algorithms
-