

# Lecture 5: Depth-First Search (DFS) & BFS vs DFS

---

## 1. Recap from Last Lecture

- We studied **Uninformed Search** basics.\
  - Understood **Breadth-First Search (BFS)**:
    - Explores nodes level by level.\
    - Guarantees shortest solution if step costs are equal.\
    - Uses **queue (FIFO)**.
- 

## 2. Depth-First Search (DFS)

- **Definition:** An uninformed search strategy that explores a branch as deeply as possible before backtracking.
- **Mechanism:**
  - Uses **stack (LIFO)** data structure (explicit or via recursion).\
  - Expands the deepest unexpanded node first.

Pseudocode (simplified):

```
function DFS(problem):
    frontier ← stack with initial state
    explored ← ∅
    while frontier not empty:
        node ← frontier.pop()
        if node is goal: return solution
        if node not in explored:
            explored.add(node)
            frontier.push(all successors of node)
```

Characteristics:

- **Completeness:** Not guaranteed (may get stuck in infinite path).\
  - **Optimality:** Not optimal (may return a longer path).\
  - **Time Complexity:**  $O(b^m)$ , where  $b$  = branching factor,  $m$  = maximum depth.\
  - **Space Complexity:**  $O(bm) \rightarrow$  very space efficient.
- 

## 3. BFS vs DFS --- Comparison Table

---

Feature BFS DFS

---

**Data Structure** Queue (FIFO) Stack (LIFO) / Recursion

**Completeness** Complete (if finite depth) Not complete (may loop forever)

**Optimality** Optimal (if uniform step Not optimal costs)

**Time**  $O(b^d)$ , where  $d$  = depth  $O(b^m)$ , where  $m$  = max Complexity of sol depth

**Space**  $O(b^d)$   $O(bm)$  (much lower memory) **Complexity**

**Best Use Case** When shortest path is When solution is deep & required space is limited

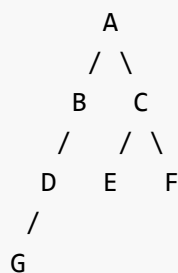
---

## 4. Visual Analogy

- **BFS:** Like ripples in water expanding outward evenly --- level by level.\
  - **DFS:** Like exploring a maze by always turning left until blocked, then backtracking.
- 

## 5. Example

Problem: Path finding from **A** → **G**



- **BFS Order:** A, B, C, D, E, F, G → Finds G at shallowest level.\
  - **DFS Order:** A, B, D, G → Finds G quickly but not guaranteed shortest.
- 

## 6. Key Takeaways

- **DFS:** Fast, memory-efficient, but risky for completeness/optimalty.\
  - **BFS:** Safe and optimal, but memory-hungry.\
  - Choice depends on **problem structure**:
    - Shallow solution → BFS.\
    - Deep solution + limited memory → DFS.
- 

## 7. Reading & Exercises

- **Reading:** AIMA, Ch. 3 (Sections on Uninformed Search, DFS, BFS).\
- **Exercise:** Implement BFS and DFS in Python/Java/C++ for a small graph (maze or tree).\
- **Discussion Question:** Can you combine BFS and DFS to balance trade-offs? (Hint: Iterative Deepening Search)

