Week 3 Lab

Introduction to *Depth First Search*

EECS 118

https://github.com/abeljim/aima-python-eecs118-fall-25

What is Depth First Search?

Depth First Search (DFS) is a graph traversal algorithm that explores as far as possible along each branch before backtracking.

Key characteristics:

- Explores **depth** before breadth
- Uses a **stack** data structure (or recursion)
- Visits nodes by going deep into the graph first
- Backtracks when no more unvisited neighbors exist

How DFS Works

DFS Algorithm

Algorithm steps:

1. Start at the root node (or arbitrary node)
2. Mark the current node as visited
3. **Explore** the first unvisited neighbor
4. Repeat step 3 recursively for each neighbor
5. **Backtrack** when no unvisited neighbors remain
6. Continue until all reachable nodes are visited

Key insight: DFS goes as deep as possible before exploring siblings

DFS Example

Tree Traversal Example

Consider traversing this tree:

```
    A
   / \
  B   C
 / \   \
D   E   F
```

DFS Traversal Order: A → B → D → E → C → F

- Start at A, go deep to B
- From B, go to D (leaf, backtrack)
- Then visit E (backtrack to B, then A)
- Finally explore C and its child F

Implementation

DFS in Python


```python
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()

    visited.add(start)

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

    return visited
```

Key Properties

Complexity & Characteristics

Complexity:
- Time: **O(V + E)**
- Space: **O(V)** for the stack/recursion

Completeness:
- Complete for finite graphs
- May get stuck in infinite paths

Optimality:
- **Not optimal** - does not guarantee shortest path
- Finds **a** solution, not necessarily the best one

When to Use DFS

Good for:

- Detecting **cycles** in graphs
- Topological sorting
- Finding **connected components**
- Maze solving and pathfinding puzzles
- Generating mazes
- Solving puzzles with **one solution** (e.g., Sudoku)

Not ideal for:

- Finding shortest paths (use BFS instead)
- When solutions are near the root

DFS vs BFS

Comparison

	DFS	BFS
Data Structure	Stack (recursion)	Queue
Traversal	Depth-first	Level-by-level
Space	O(h) - height	O(w) - width
Optimal	No	Yes (unweighted)
Use Case	Cycle detection	Shortest path

Summary

Remember:

- DFS explores **deep** before wide
- Uses recursion or explicit stack
- $O(V + E)$ time complexity
- Great for cycle detection and topological sorting
- Not optimal for shortest paths

Practice: Implement DFS on the graph problems in the lab repository!

Questions?

“https://github.com/abeljim/aima-python-eecs118-fall-25/lab_plans/week3.pdf”