Lecture 12 - Graphs, DFS, and BFS

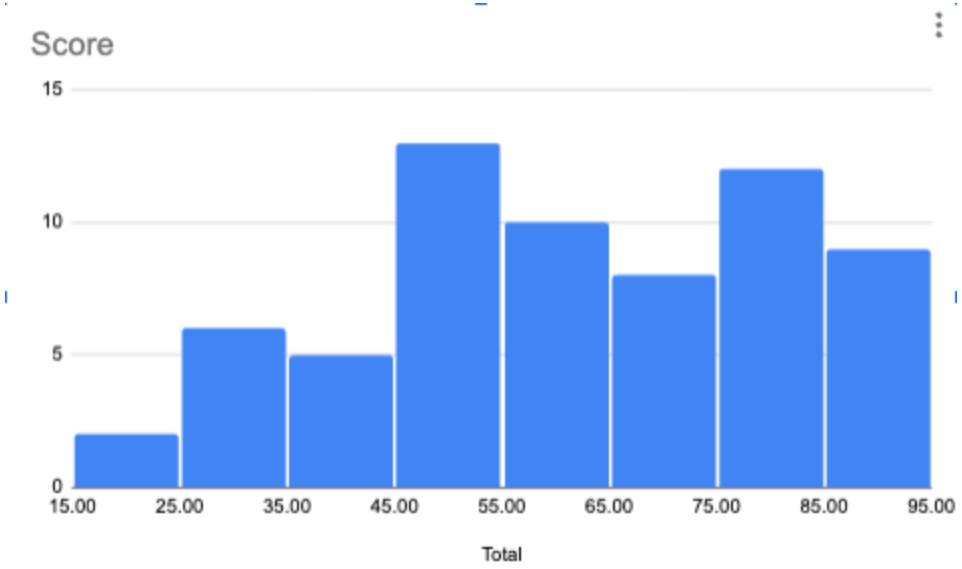
Fall 2025, Korea University

Instructor: Gabin An (gabin_an@korea.ac.kr)

HAPPINESS IS

... having finished all of your exams!

Midterm Exam



- Average: 61 (std: 20.5)
- Median: 59
- Please come see me after class if you'd like to check your exam paper.

Course Outline (After Midterm)

- Part 3: Data Structures - Continued
  - **Graphs, Graph Search and Applications** 👈
- Part 4: Dynamic Programming
  - Shortest-Path: Dijkstra, Bellman-Ford, Floyd-Warshall Algorithms
  - More General DP: Longest Common Subsequence, Knapsack Problem
- Part 5: Greedy Algorithms and Others
  - Scheduling Problem, Optimal Codes
  - Minimum Spanning Trees
  - Max Flow, Min Cut and Ford-Fulkerson Algorithms
  - Stable Matching, Gale-Shapley Algorithm

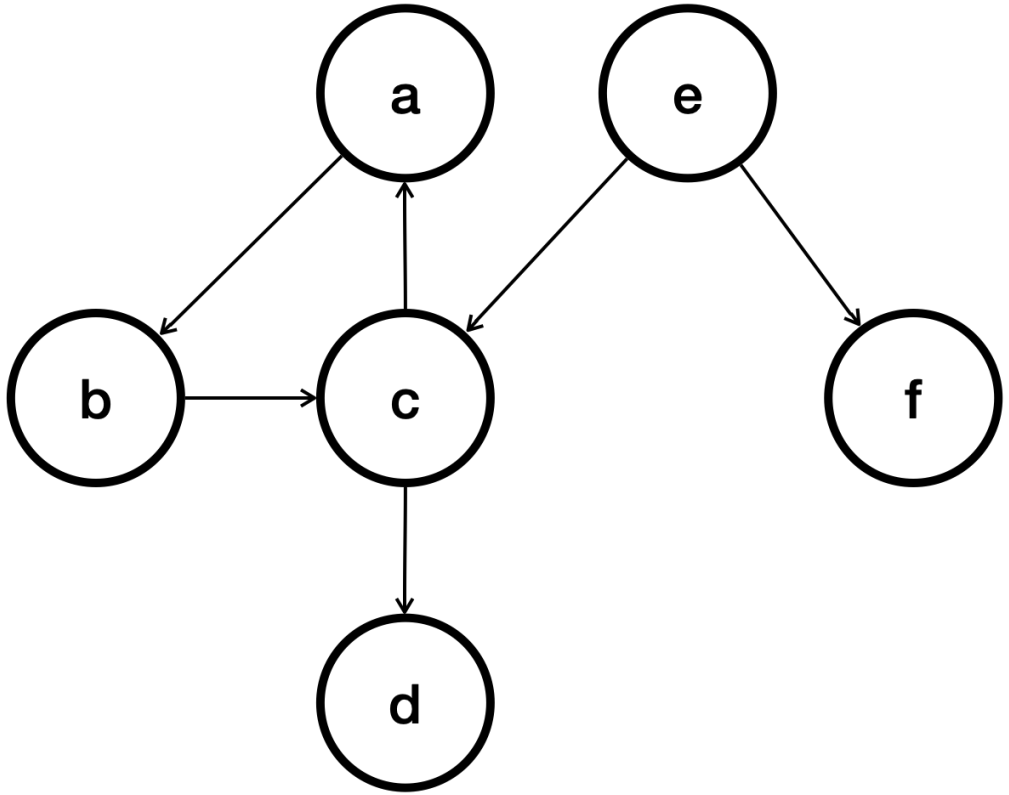
Today's Objectives

- Learn the basics of **graphs**, including terminology like **vertices**, **edges**, and types of graphs.
- Understand different methods for representing graphs in memory: Adjacency Matrix and Adjacency List.
- Explore **Depth First Search (DFS)**
- Explore **Breadth First Search (BFS)**
- Compare and contrast DFS and BFS to understand their core differences.

What are Graphs?

- A graph $G = (V, E)$ is a set of vertices (nodes) V and edges E connecting them.
- n = number of vertices ($|V|$), m = number of edges ($|E|$).
- **Directed vs. Undirected:**
  - **Undirected:** edges are bidirectional. $(i, j) \in E \iff (j, i) \in E$.
  - **Directed:** edges have a direction.
- **Connected Graph:** A path exists between any two nodes.
- **Sparse vs. Dense:**
  - **Sparse:** Few edges, e.g., $m = \Theta(n)$.
  - **Dense:** Many edges, e.g., $m = \Theta(n^2)$.

Example: Directed Graph w/ Six Nodes and Six Edges

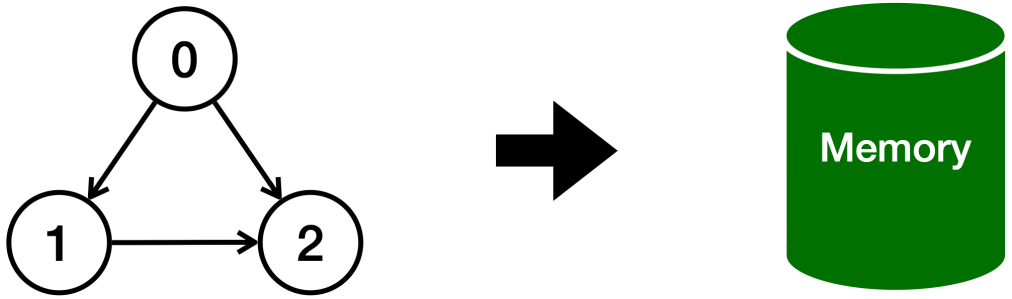


Degree of a Node

- The number of edges connected to that node.
- For undirected graphs:
  - $deg(u) = \text{number of edges incident to } u$
- For directed graphs:

  Each node has two types of degrees:
  - In-degree: number of incoming edges (edges ending at the node)
  - Out-degree: number of outgoing edges (edges starting from the node)

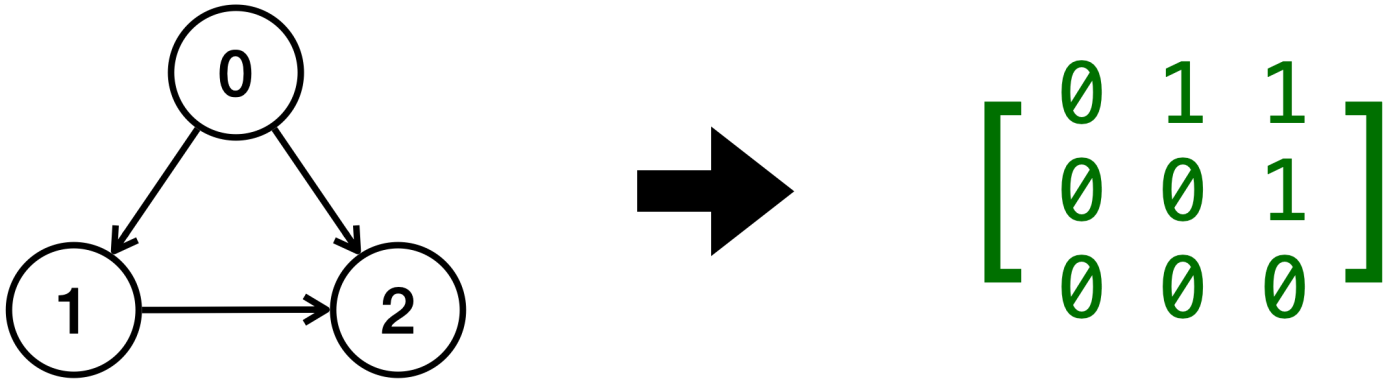
Representing Graphs



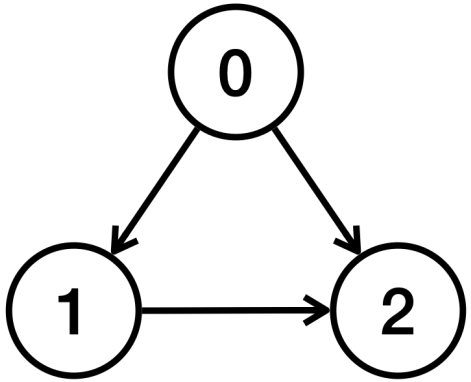
- How to store a graph in memory? Two main methods:
    i. **Adjacency Matrix**
    ii. **Adjacency List**

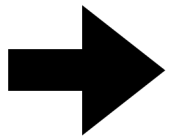
Representing Graphs - 1. Adjacency Matrix

- An $n \times n$ binary matrix.
- Matrix element (i, j) is **1** if edge (i, j) exists, **0** otherwise.



Representing Graphs - 1. Adjacency Matrix - Continued



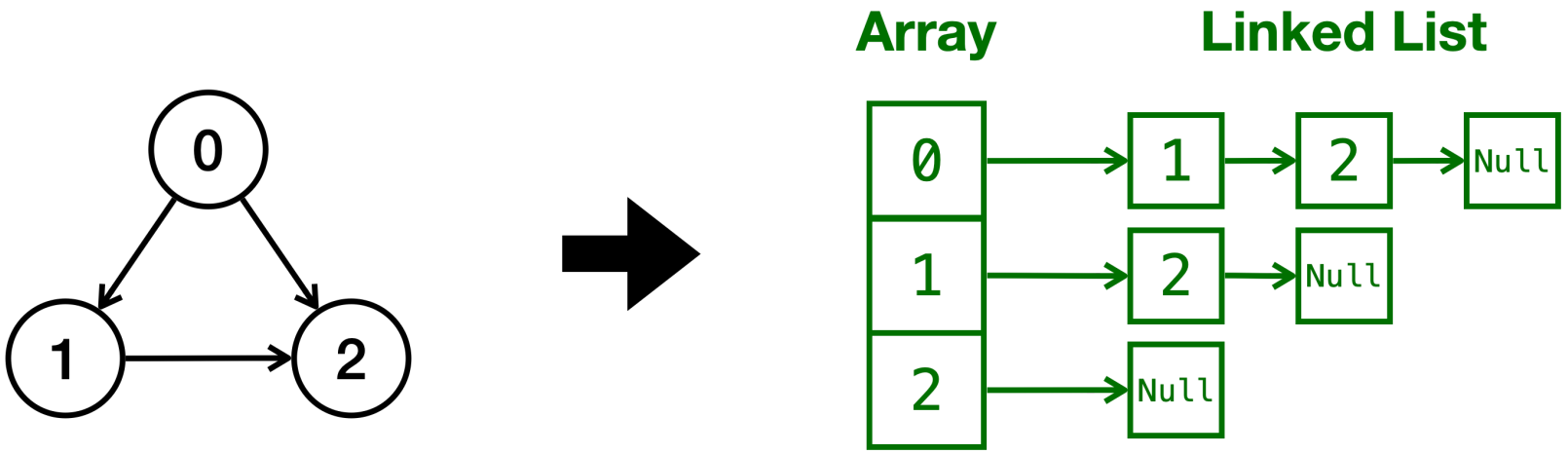


$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

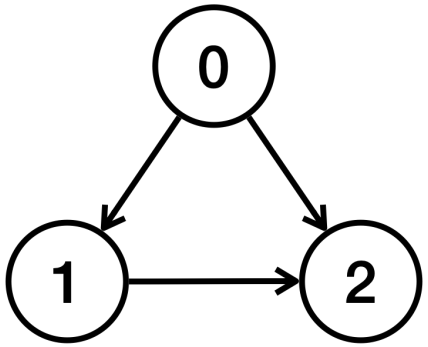
- **Space:** $\Theta(n^2)$ (why? $n \times n$).
- **Edge Lookup:** $O(1)$ time to check if an edge exists.
- **Neighbor Enumeration:** $\Theta(n)$ time.

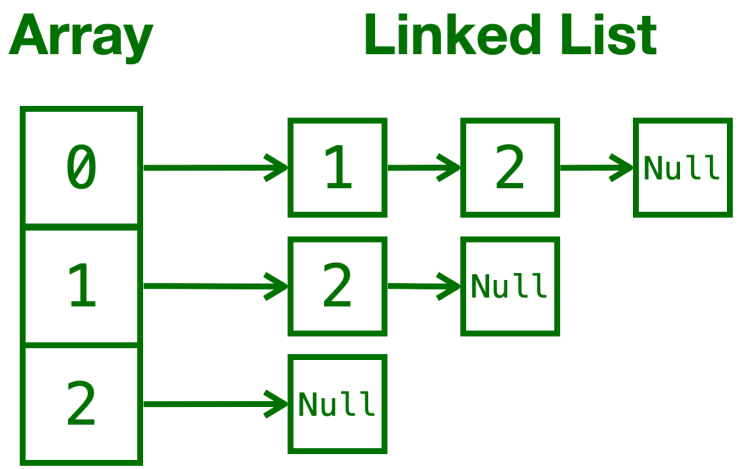
Representing Graphs - 2. Adjacency List

- An array A of n lists.
- $A[u]$ contains a list of all vertices v where $(u, v) \in E$.



Representing Graphs - 2. Adjacency List - Continued





- **Space:** $\Theta(n+m)$.
- **Edge Lookup:** $O(\deg(u))$ time.
- **Neighbor Enumeration:** $O(\deg(u))$ time.

Representing Graphs - Summary

Representation	Space Complexity	Check if edge (u, v) exists	List all neighbors of u	Notes
Adjacency Matrix	$\Theta(n^2)$	$\Theta(1)$	$\Theta(n)$	Fast edge lookup but uses lots of memory.
Adjacency List	$\Theta(n+m)$	$O(deg(u))$ (≈ $O(V)$ worst case)	$\Theta(deg(u))$	Efficient for sparse graphs.

Graph Traversal Algorithms

- Depth-First Search (DFS)
- Breadth-First Search (BFS)

Depth First Search (DFS)

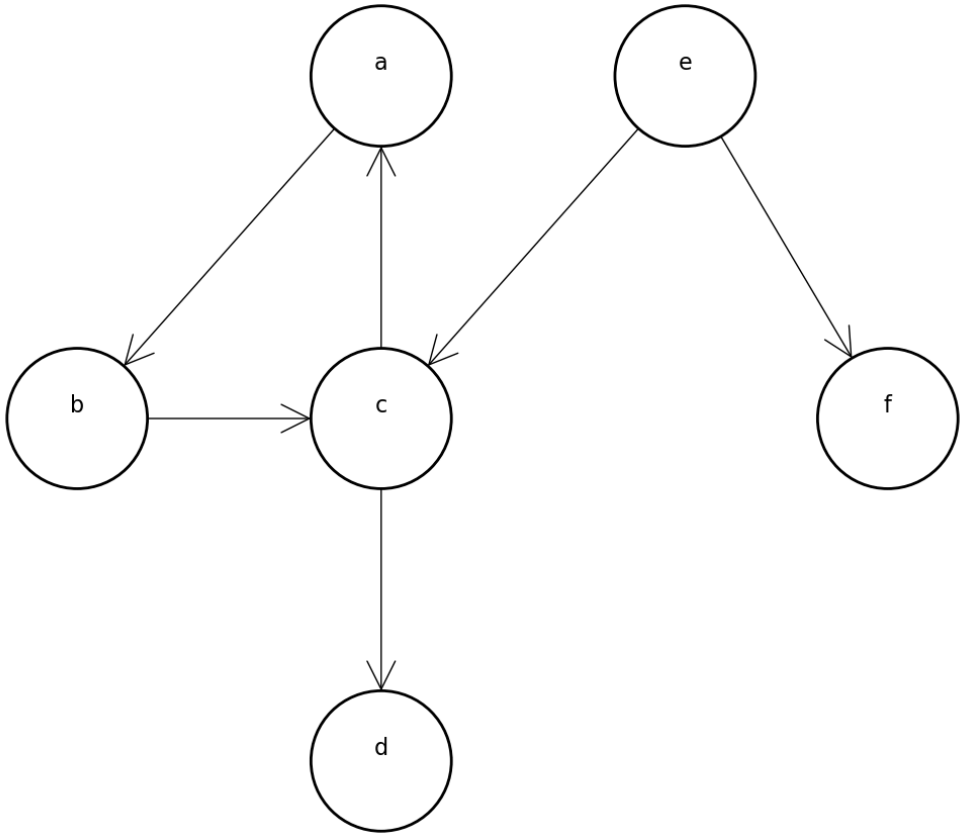
- **Strategy:** Explores a path **as far as possible** before backtracking.
- **DFS Augmentations (to prevent loops):**
  - **Colors:**
    - **White:** Unvisited.
    - **Gray:** Visited, but not all neighbors explored.
    - **Green :** Completely processed.
  - **Parent Pointers:** Keeps track of the traversal path.
  - **Timestamps:**
    - **Discovery time ($d(v)$):** When a node is first visited.
    - **Finish time ($f(v)$):** When a node is completely processed.

DFS Pseudocode

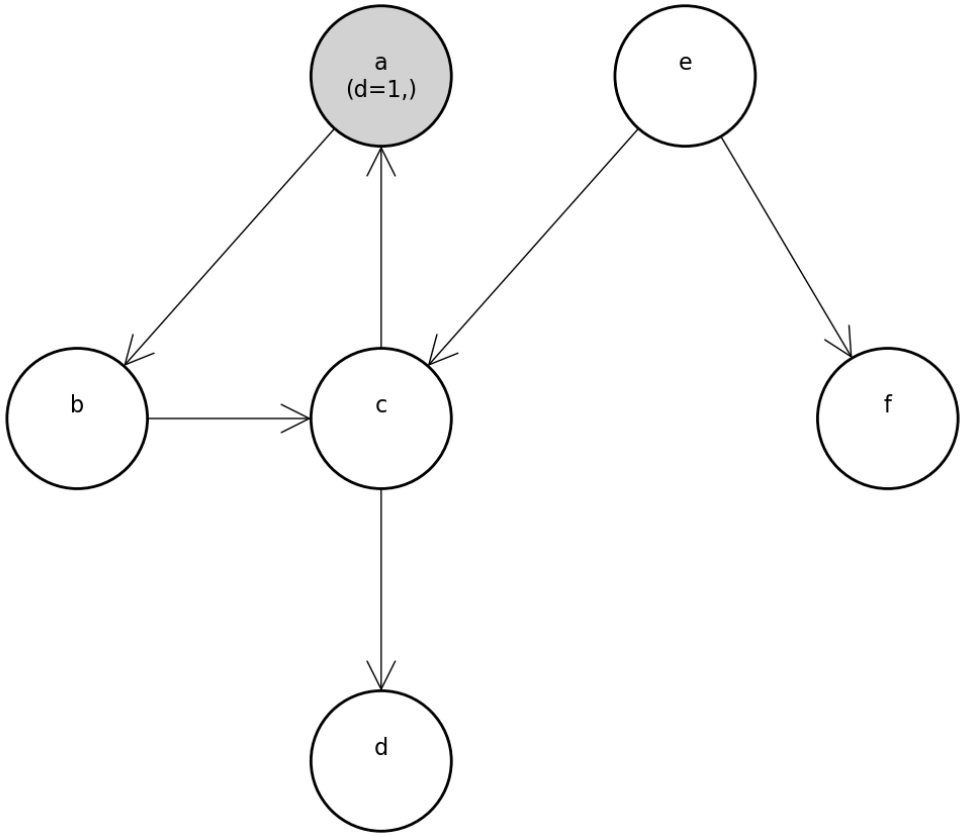
- Initially, every node is marked as unvisited (`white`).

```
Algorithm DFS(s, t):
    color(s) ← gray
    d(s) ← t
    t ← t + 1
    foreach v in N(s) do
        if color(v) == white then
            p(v) ← s
            t ← DFS(v, t)
            t ← t + 1
    f(s) ← t
    color(s) ← green
    return f(s)
```

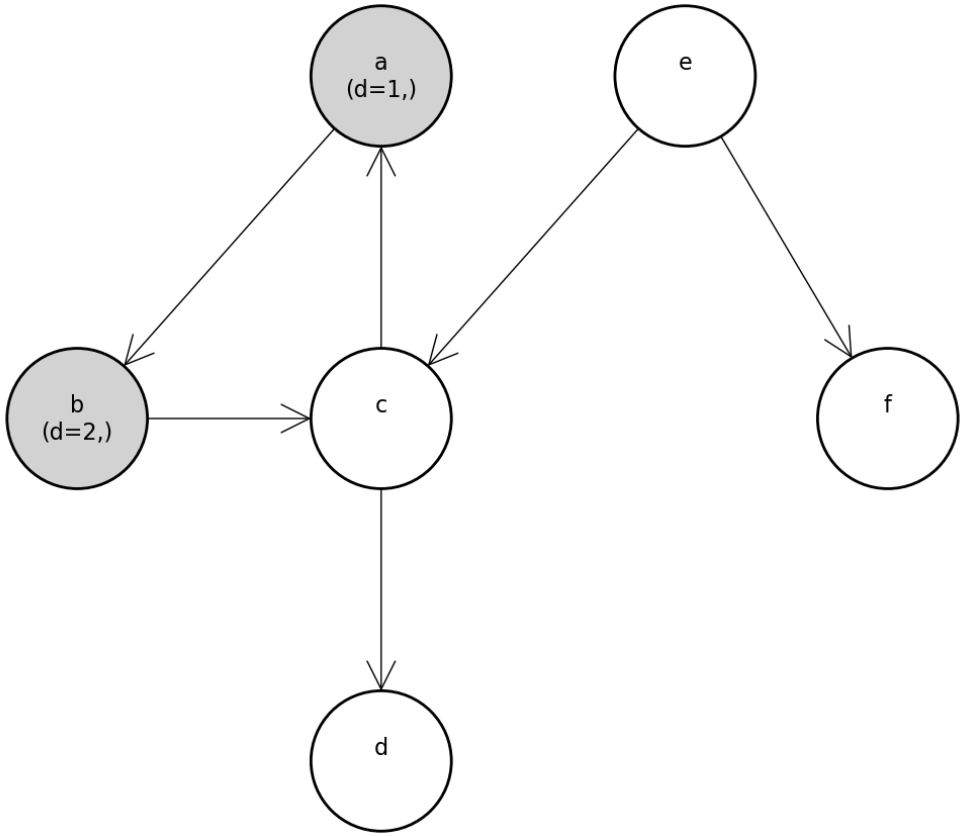
DFS Example



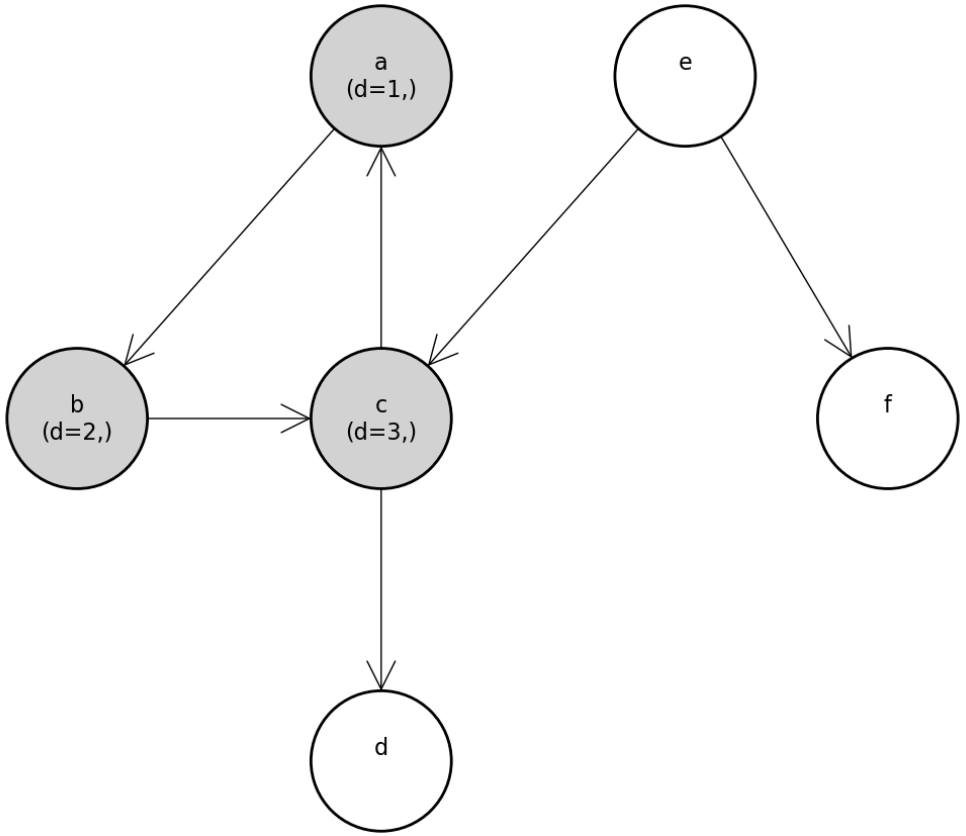
- Start at a white node `a`.



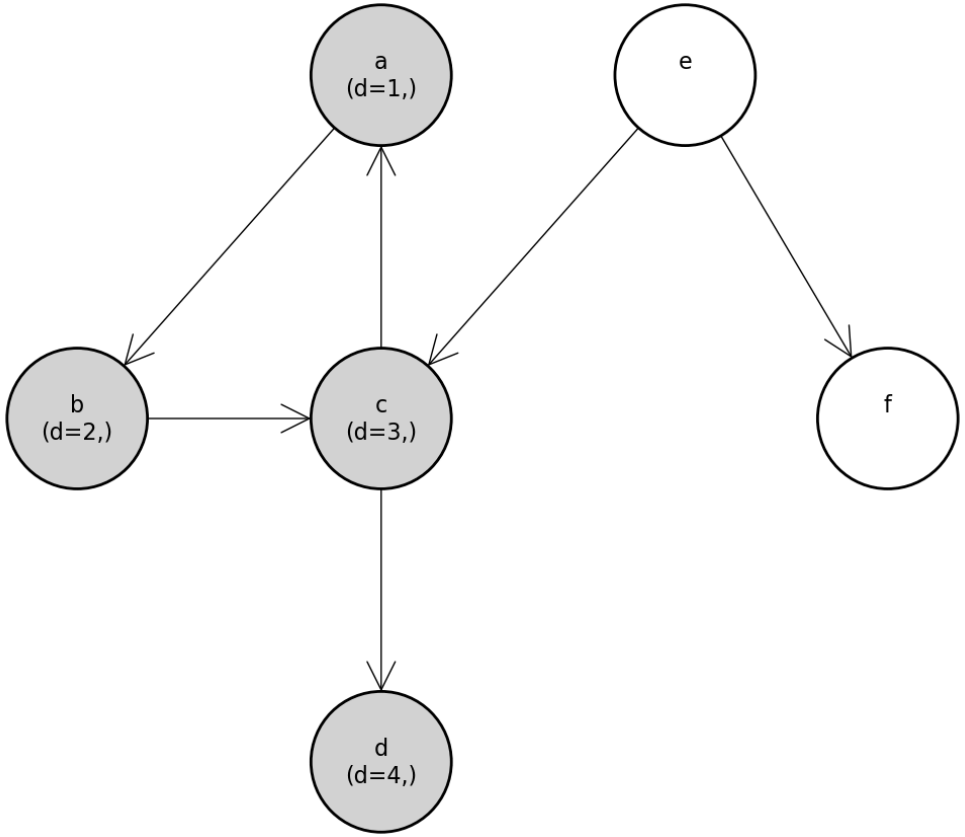
- From node `a`, we will visit all of `a`'s children, namely node `b`.



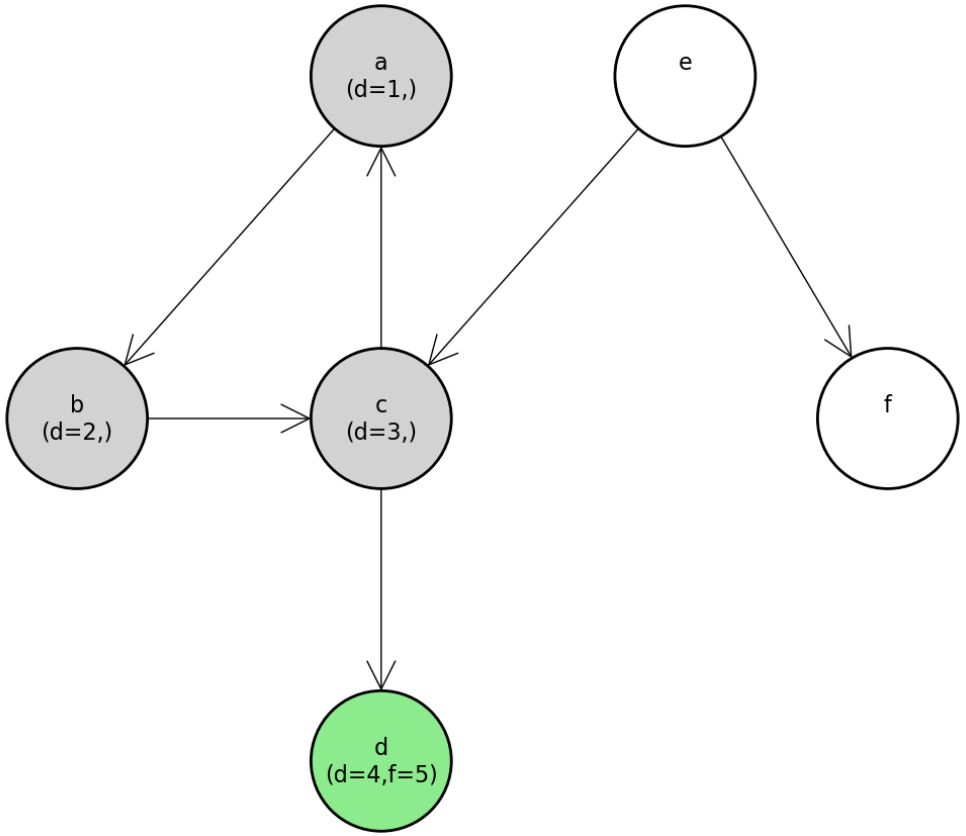
- We now visit `b`'s child, node `c`.

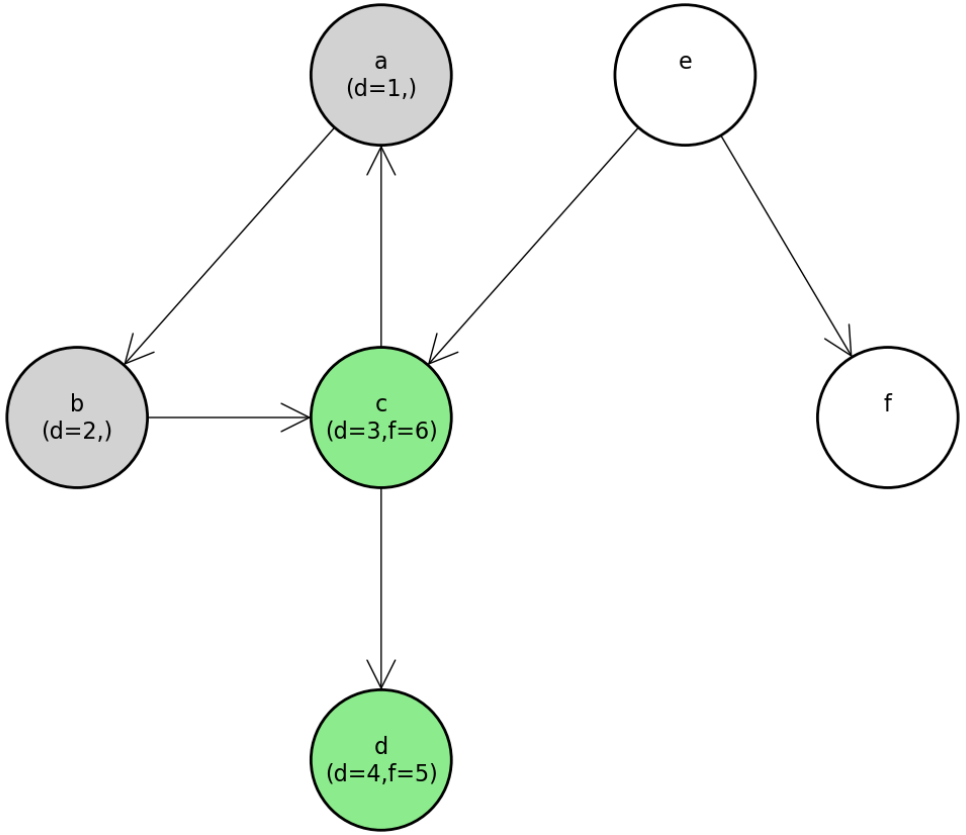


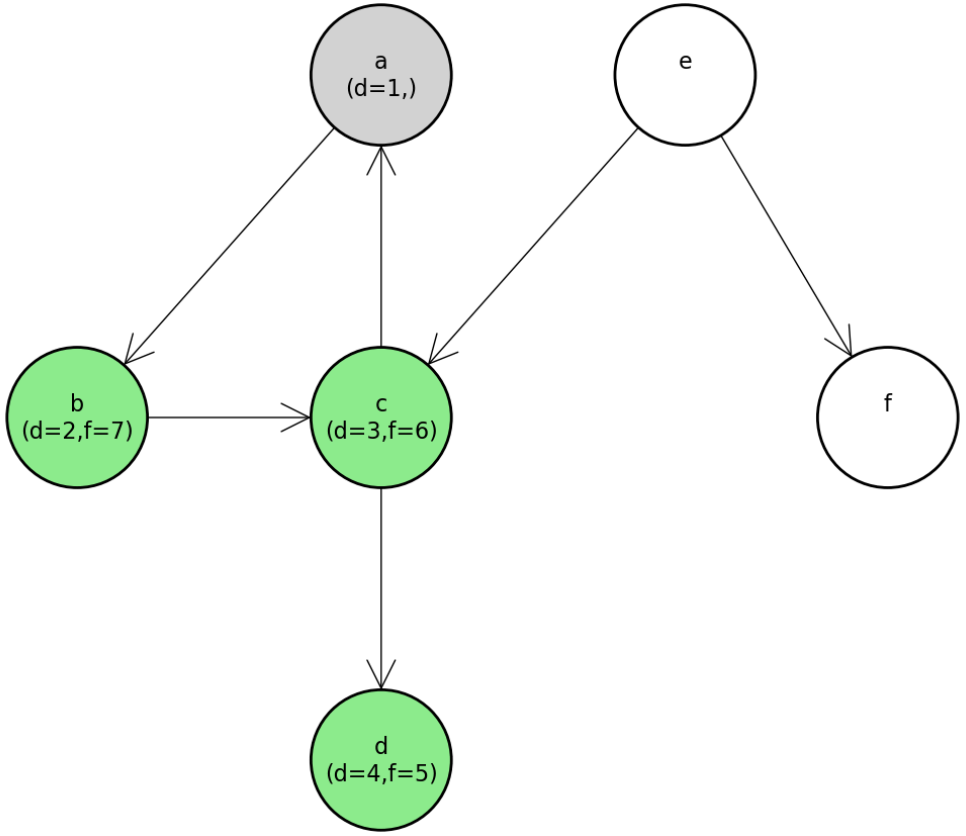
- We will next search `c`'s second child, node `d`, as another child, node `a`, has already been visited (i.e., colored gray).

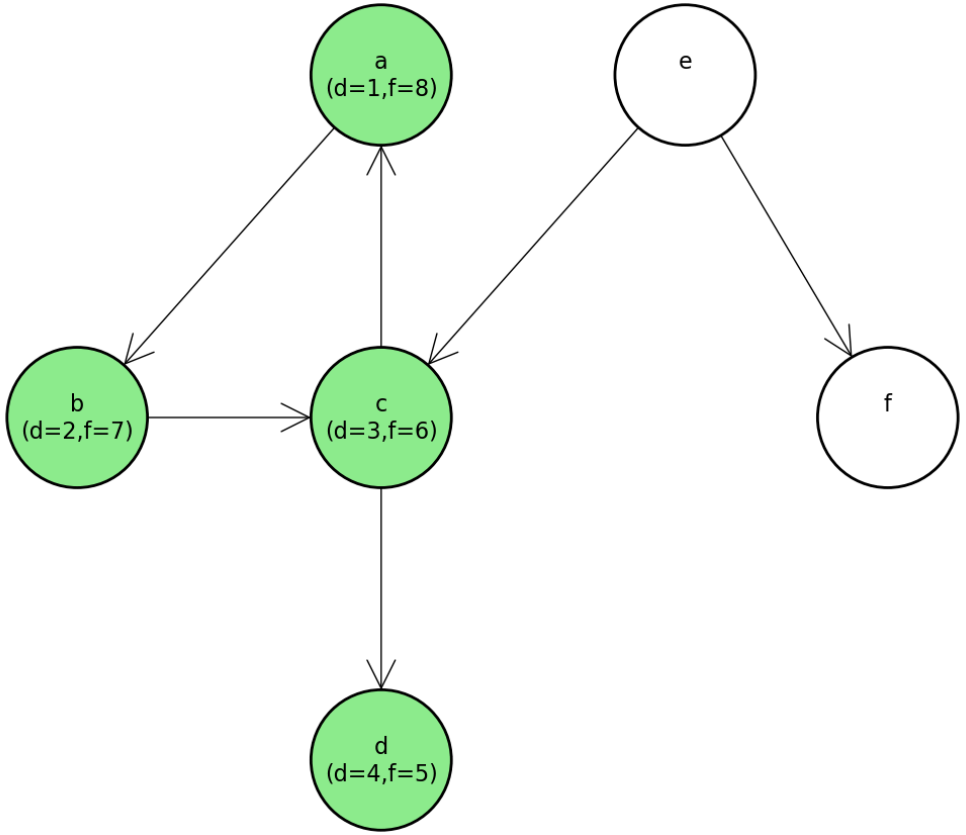


- Since node `d` has no children, we return back to its parent node, `c`, and continue to go back up the path we took, marking nodes with a finish time when we have searched all of their children.

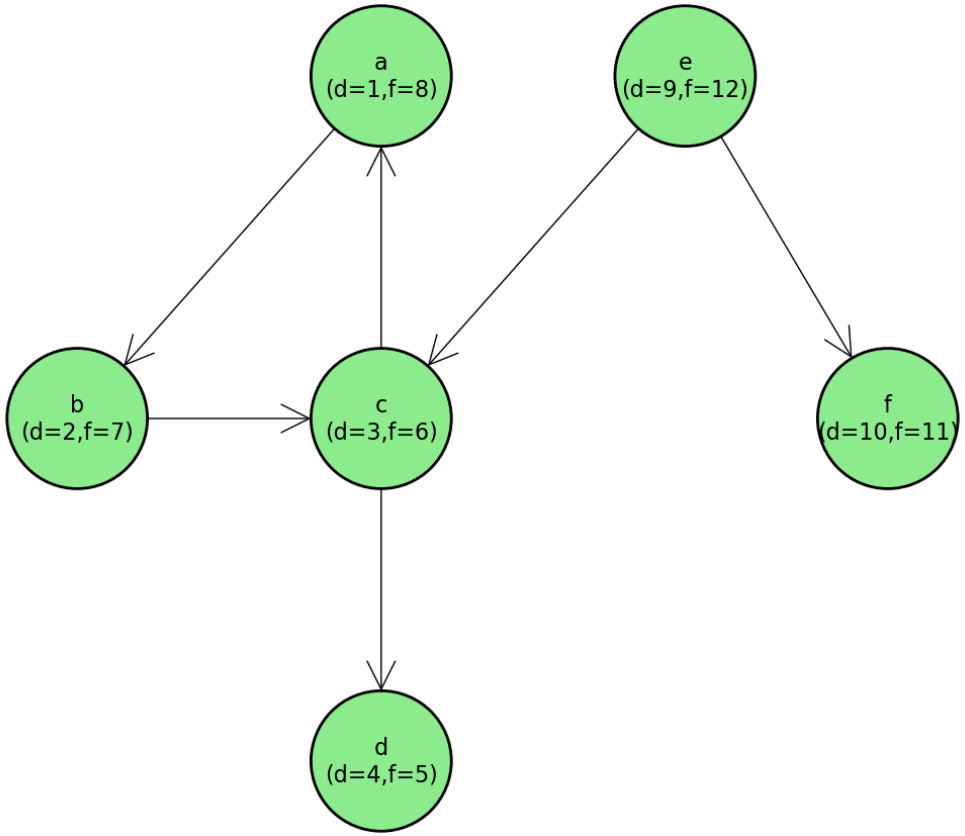



a
(d=1,)
e
b
(d=2,)
c
(d=3,f=6)
f
d
(d=4,f=5)


a
(d=1,)
e
b
(d=2,f=7)
c
(d=3,f=6)
f
d
(d=4,f=5)


a
(d=1,f=8)
e
b
(d=2,f=7)
c
(d=3,f=6)
f
d
(d=4,f=5)

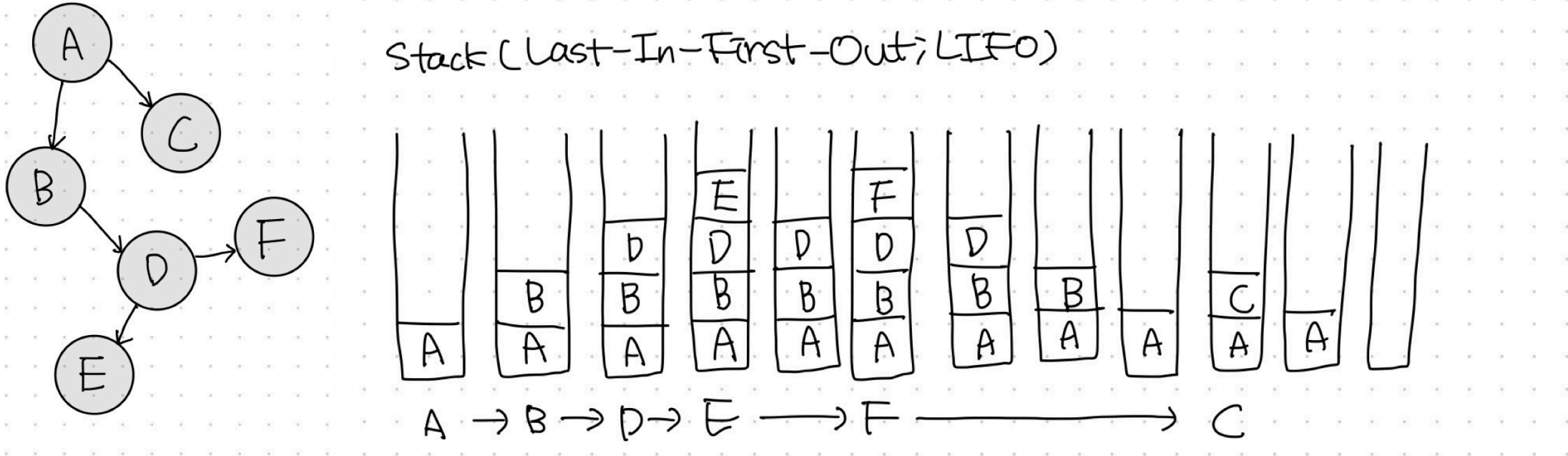
- We start with a new source node `e` and run DFS to completion.



DFS Runtime

- **Runtime:** $O(\#\text{ of node visits} + \#\text{ of edge scans})$.
- Each node is visited at most once, so node visits are $O(n)$.
- Each edge is scanned at most twice (or once for directed graphs), so edge scans are $O(m)$.
- **Total Runtime:** $O(m + n)$.

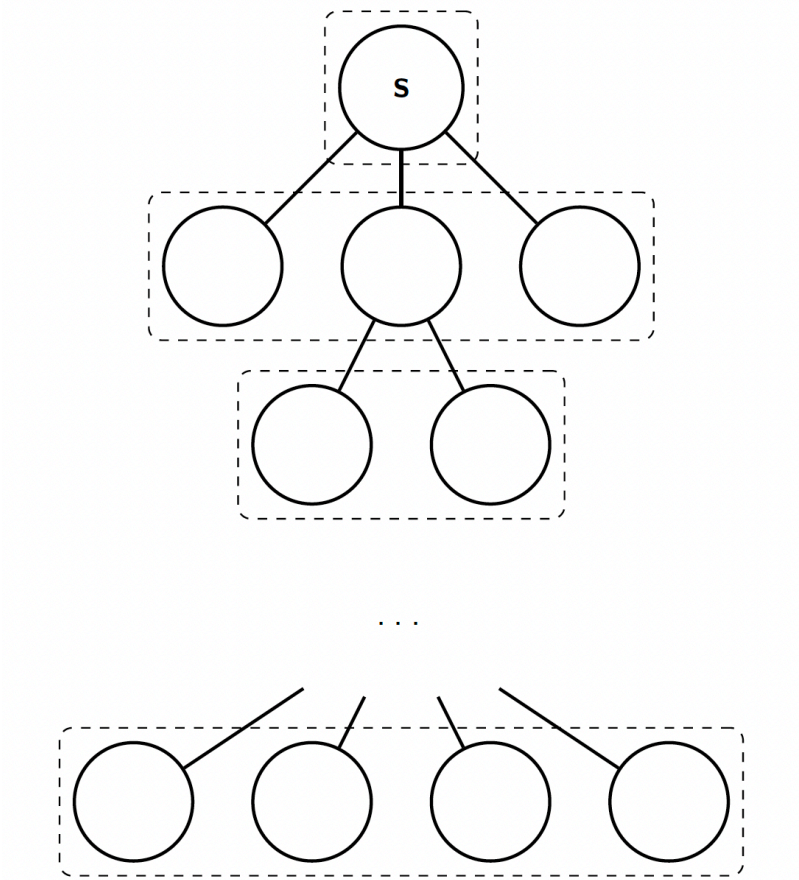
DFS and Stack



Breadth First Search (BFS)

- **Strategy:** Expands the search frontier uniformly, visiting all nodes at distance k before moving to nodes at distance $k + 1$.
- **Data Structure:** A **queue** (First-In, First-Out) is used to explore nodes in order of their distance from the source.
- **Level Sets (L_i):** BFS computes sets L_i containing all nodes at distance i from the source.

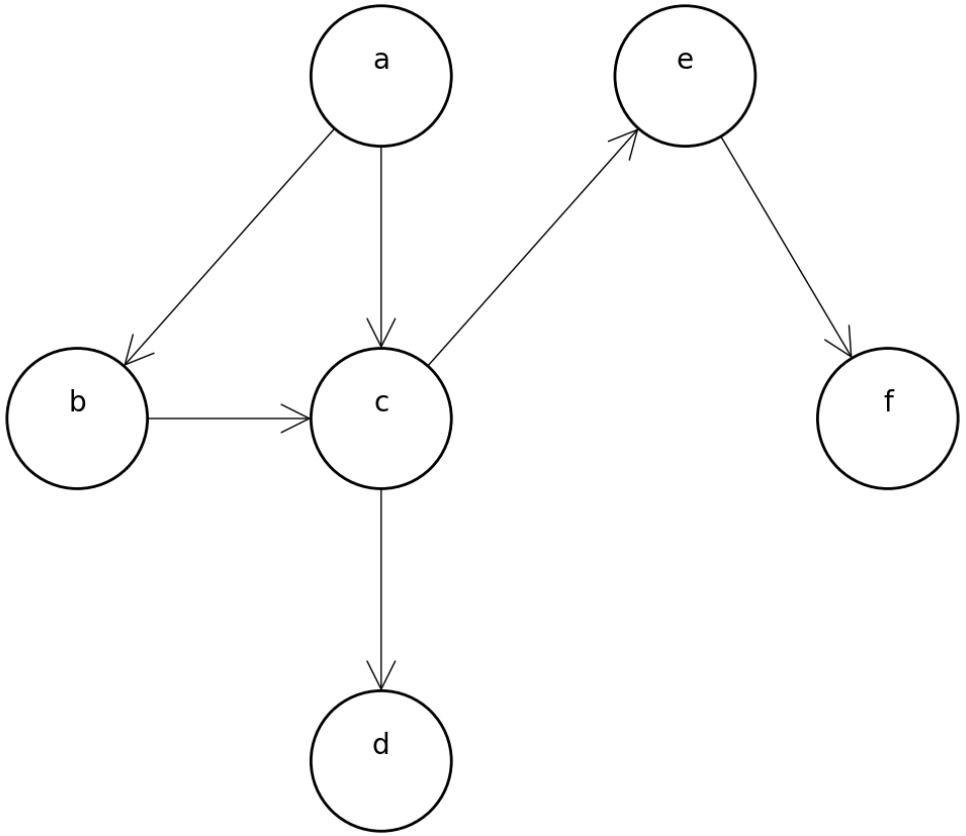
Level Sets?



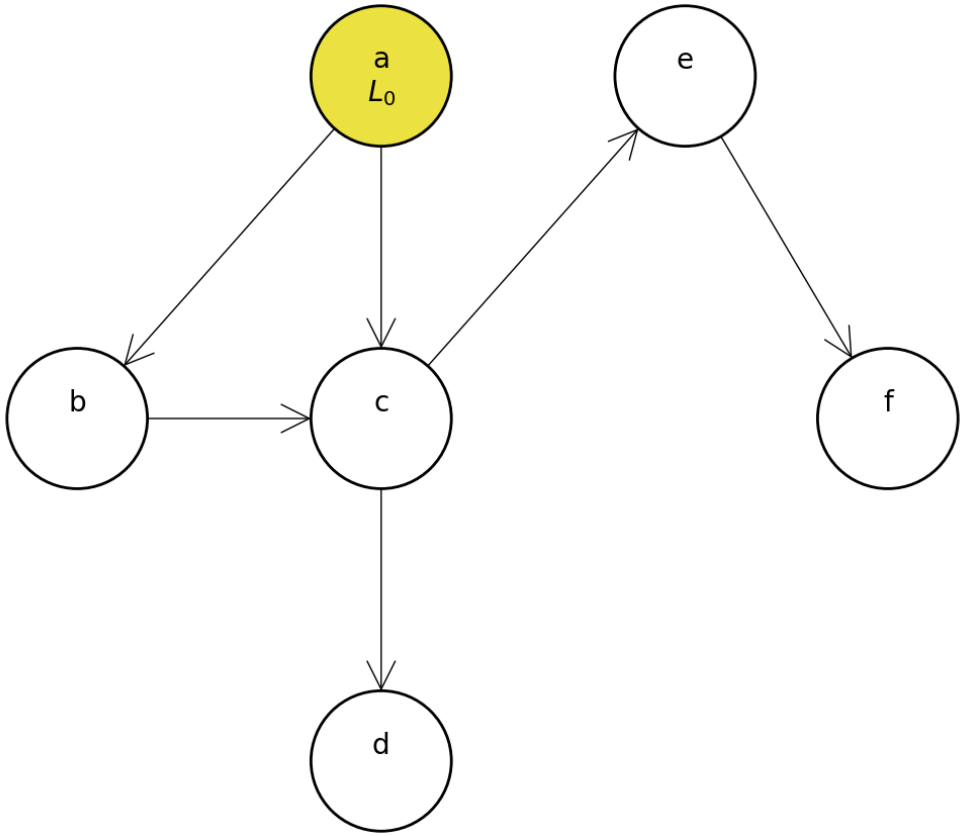
BFS Pseudocode

```
Algorithm BFS(s):
    vis[v] ← false for all v
    L_0 ← {s}
    vis[s] ← true
    for i = 0...n-1 do
        if L_i is empty then exit
        while L_i is not empty do
            u ← L_i.pop()
            foreach x in N(u) do
                if vis[x] == false then
                    vis[x] ← true
                    L_{i+1}.insert(x)
                    p(x) ← u
```

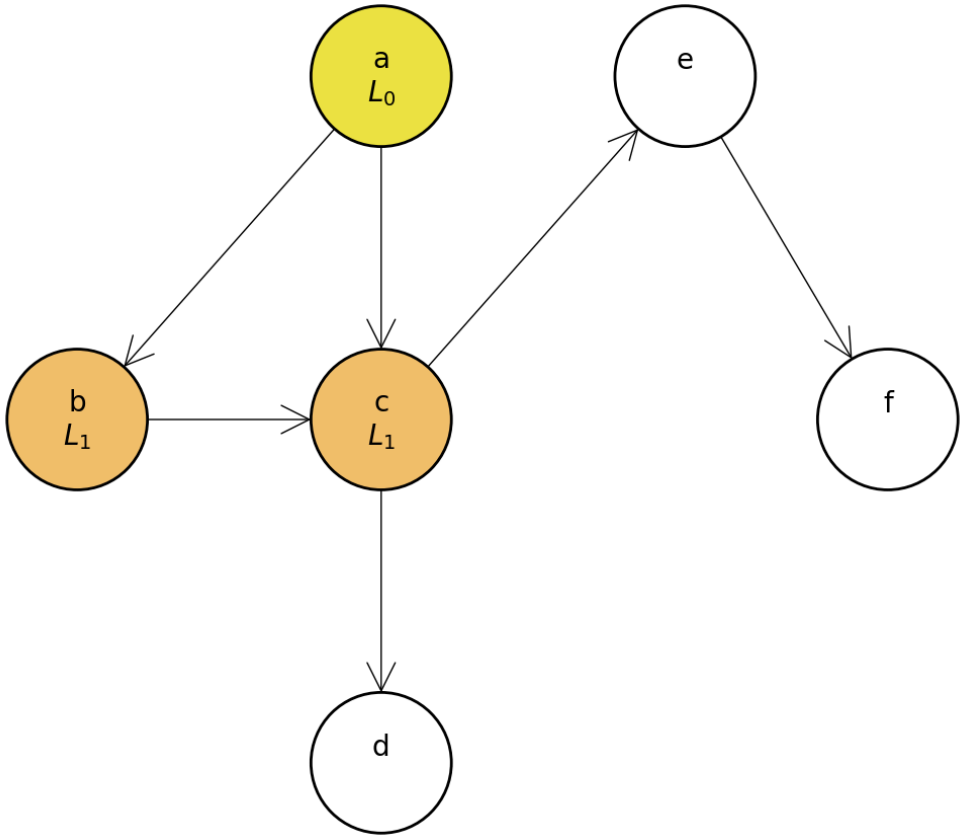
BFS Example



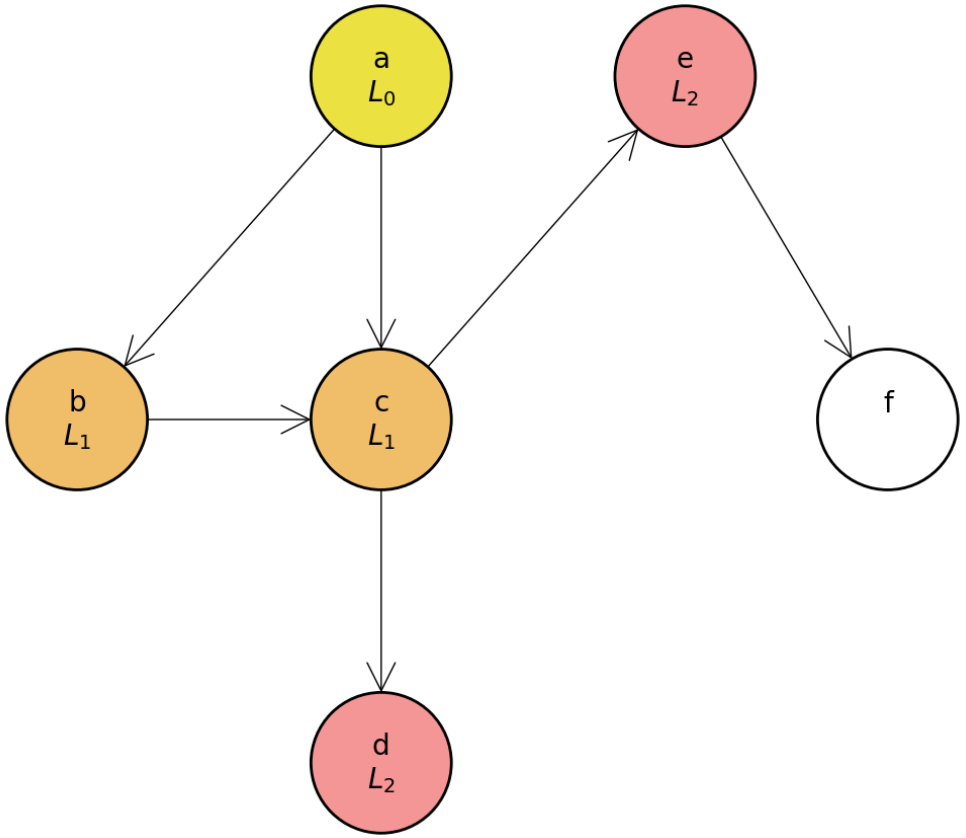
- We begin with the first level set, L_0, which contains only the source node `a`.



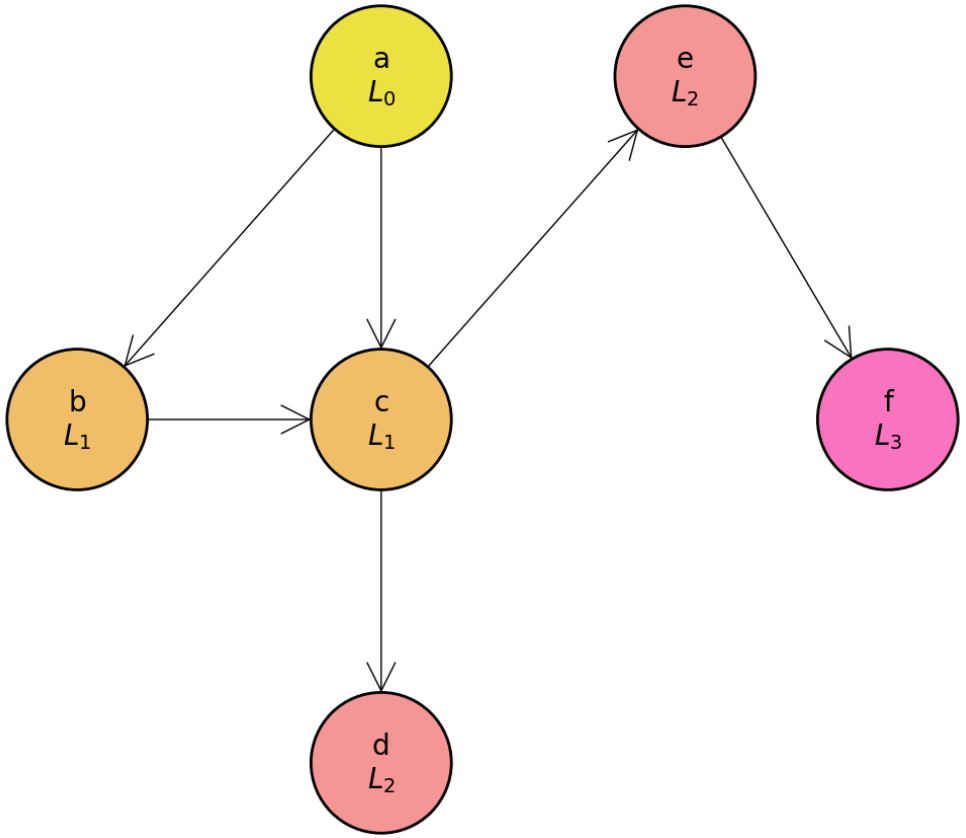
- Next, we explore the second level set, L_1, consisting of nodes directly reachable from L_0.



- We then move on to the third level set, L_2, which contains nodes directly reachable from L_1.



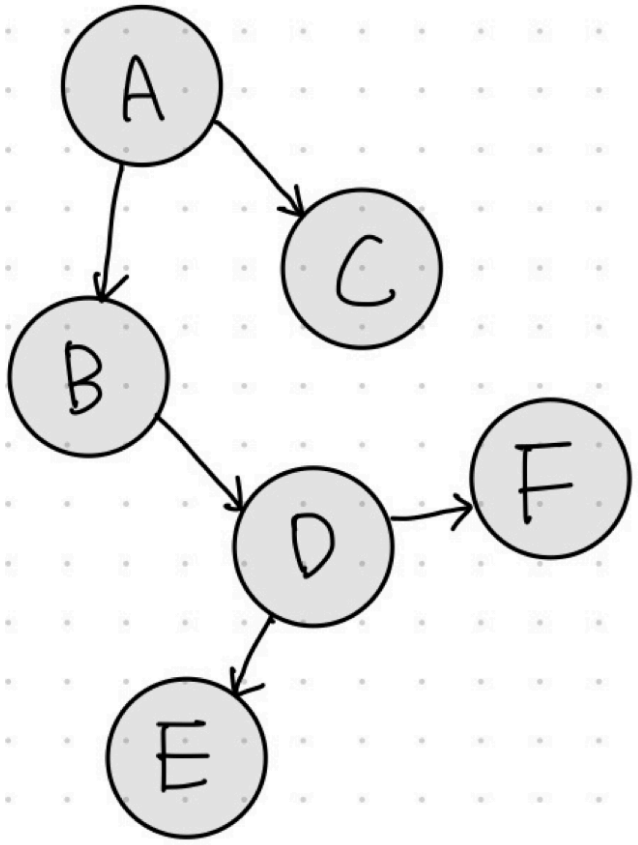
- Finally, we reach the last level set, L_4. With no further children to explore, the search terminates here.



BFS Runtime

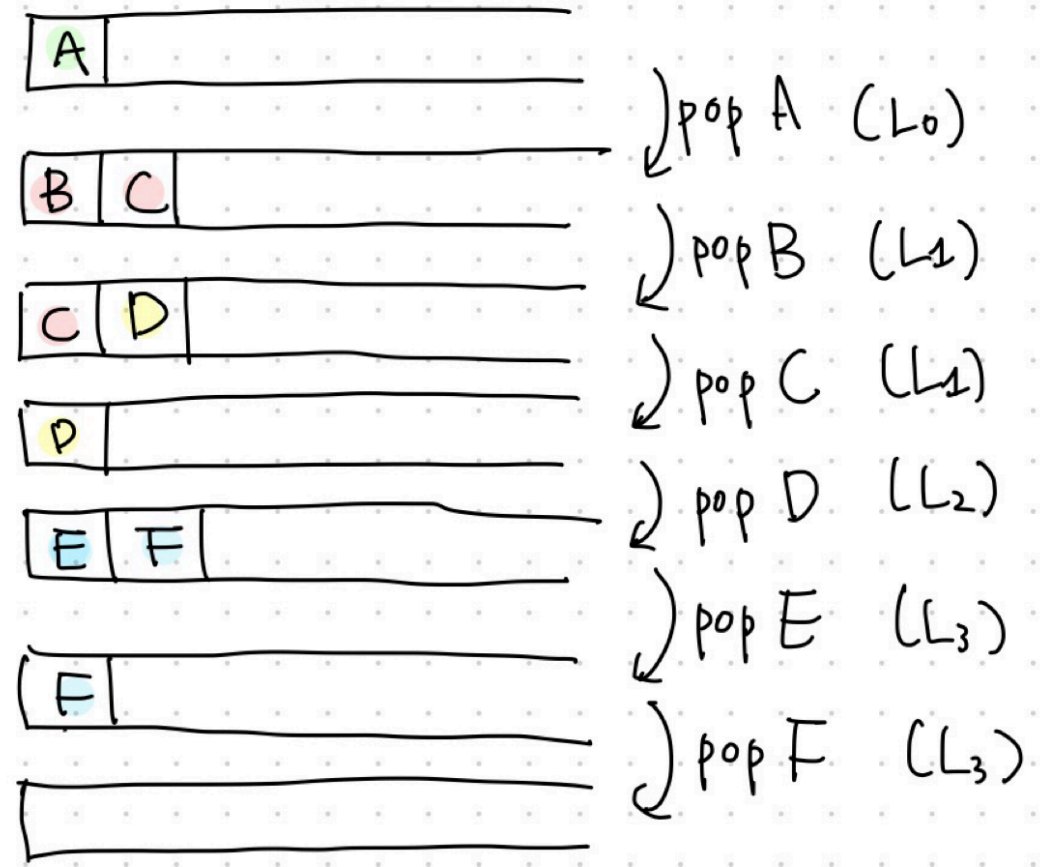
- **Runtime:** $O(\#\text{ nodes visited} + \#\text{ edges scanned})$.
- Each node is visited once, so node visits are $O(n)$.
- Each edge is scanned once, so edge scans are $O(m)$.
- **Total Runtime:** $O(m + n)$.

BFS and Queue



BFS

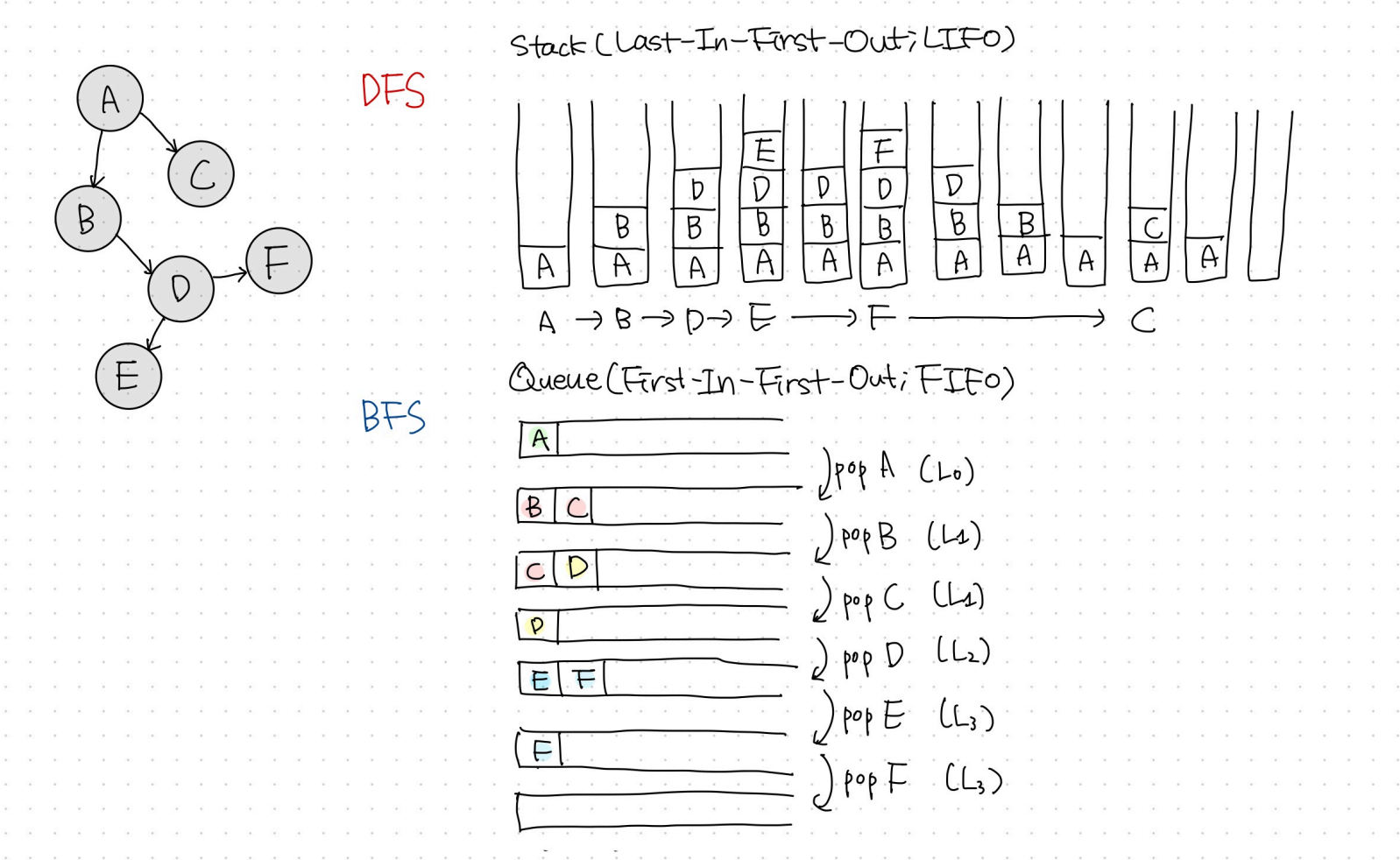
Queue (First-In-First-Out; FIFO)



BFS vs. DFS Simplified

- **DFS:** Explore as deep as possible before backtracking
    - Uses a **stack** to go as far down a path as possible before backtracking. It's like exploring a single winding tunnel to its end before coming back.
- **BFS:** Explore all neighbors level by level
    - Uses a **queue** to explore nodes closest to the source first. It's like ripples spreading out in a pond.

BFS vs. DFS Simplified



Credits & Resources

Lecture materials adapted from:

- Stanford CS161 slides and lecture notes
  - https://stanford-cs161.github.io/winter2025/
- *Algorithms Illuminated* by Tim Roughgarden
  - https://algorithmsilluminated.com/