Artificial Intelligence in Java

RP by Rodrigo Martins Pagliares

Computer Science Department - UNIFAL – Universidade Federal de Alfenas - MG - Brazil

Last update: 04/29/2025

Breadth-First Search (BFS)

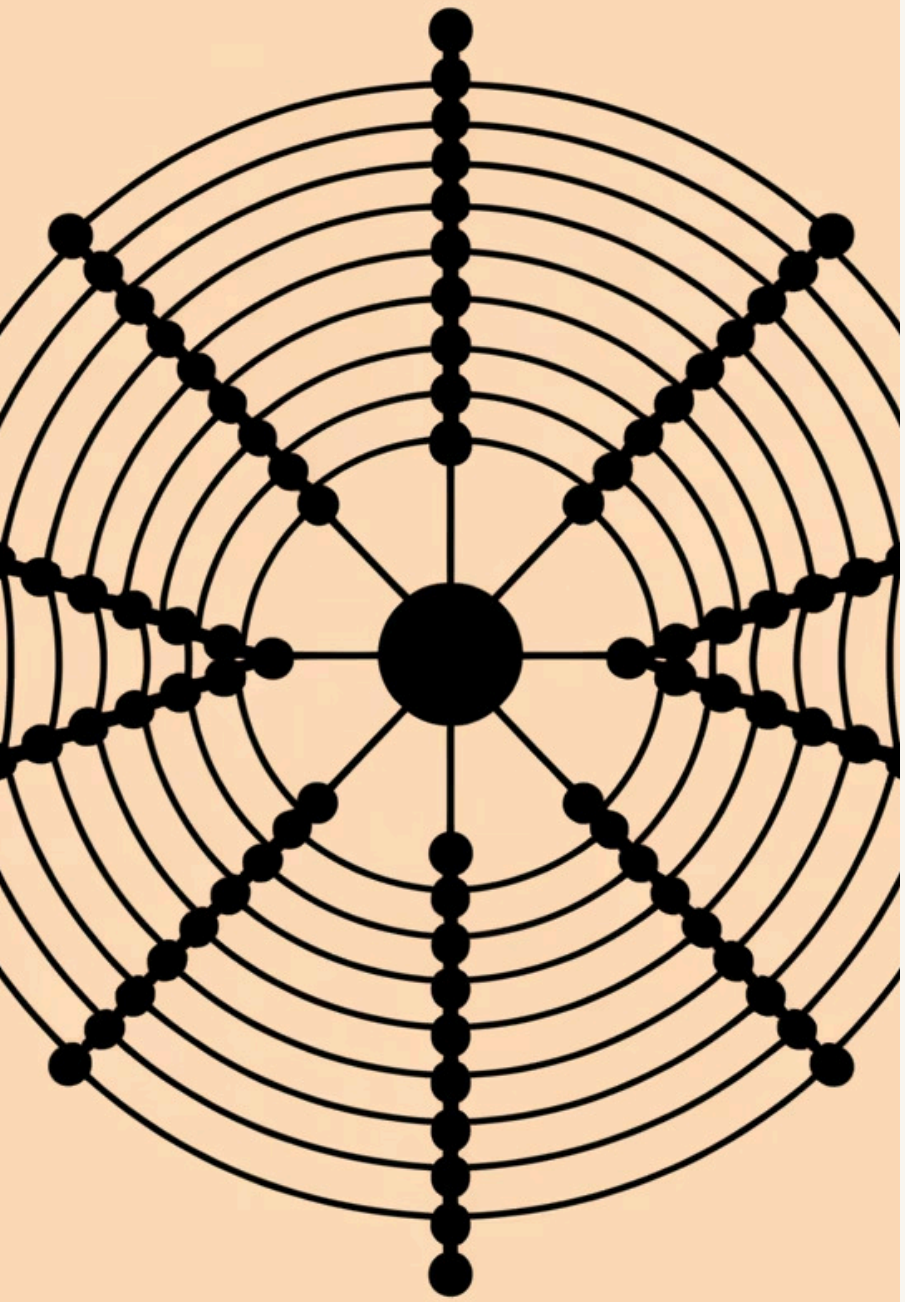
Why study **graph traversal** in **AI**?

Many **AI problems can be reduced** to **graph traversal** and **pathfinding**

BFS and **DFS** help solve **AI** problems efficiently

This chapter focuses on the **Breadth-First Search algorithm**

RP **by Rodrigo Martins Pagliares**



What is Breadth-First Search?

Layer by Layer Traversal

BFS traverses a graph **layer by layer**

Complete Coverage

Visits every vertex exactly once

Neighbor First Approach

Expands neighbors of the current vertex before going deeper

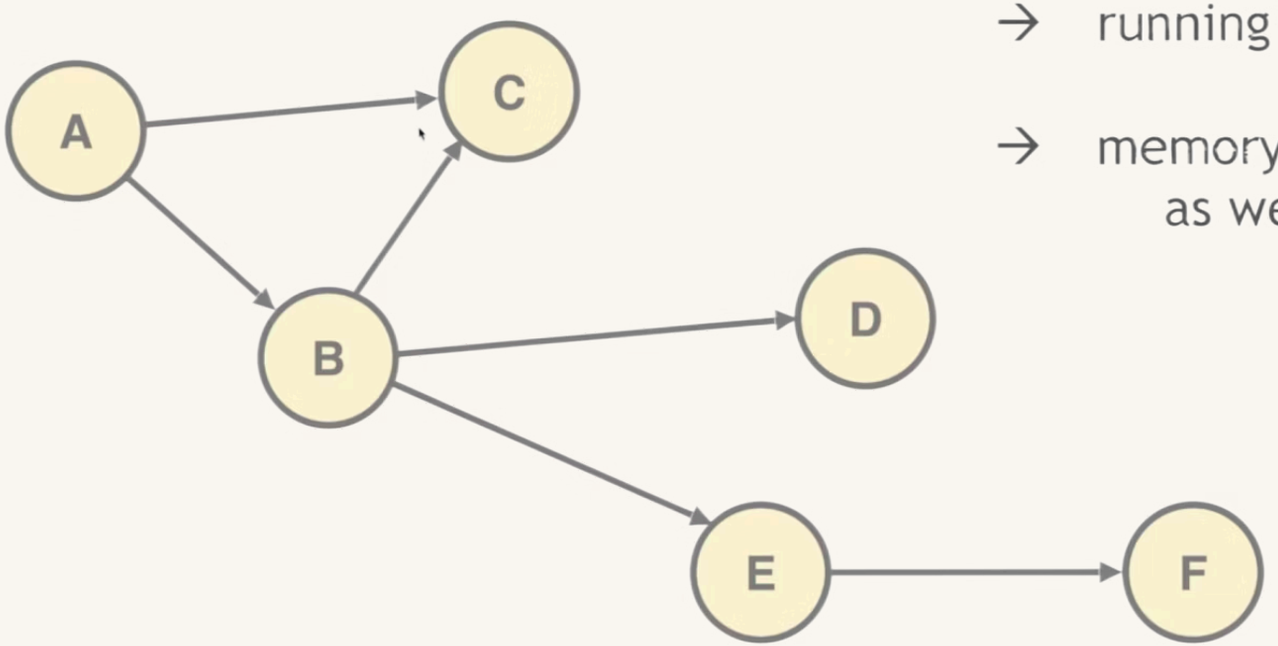
Path Finding

Useful for **finding shortest paths** in **unweighted graphs**

Breadth-First Search

There is a **G(V,E)** graph and the aim is to visit every **V** vertex exactly once

Breadth-first search visits the neighbors and then the neighbors of these new vertices until all nodes are visited



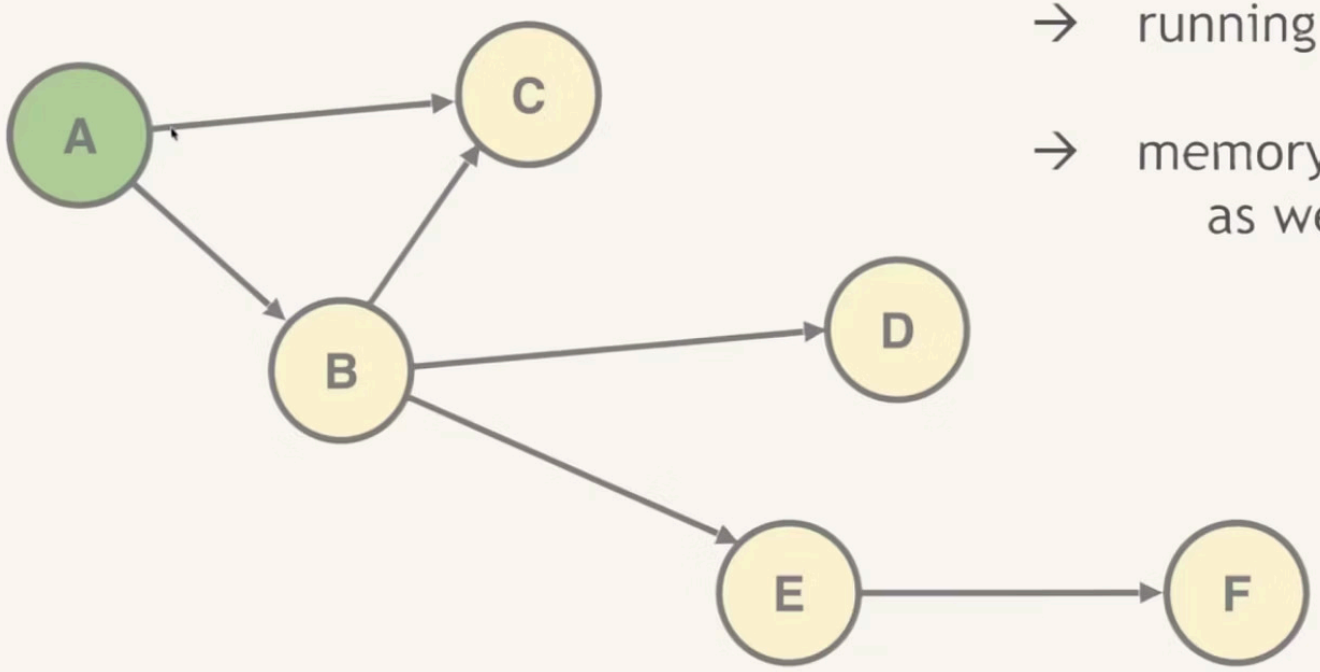
→ running time complexity is **O(V+E)**

→ memory complexity is not that favorable as we have to **store lot of references**

Breadth-First Search

There is a **G(V,E)** graph and the aim is to visit every **V** vertex exactly once

Breadth-first search visits the neighbors and then the neighbors of these new vertices until all nodes are visited



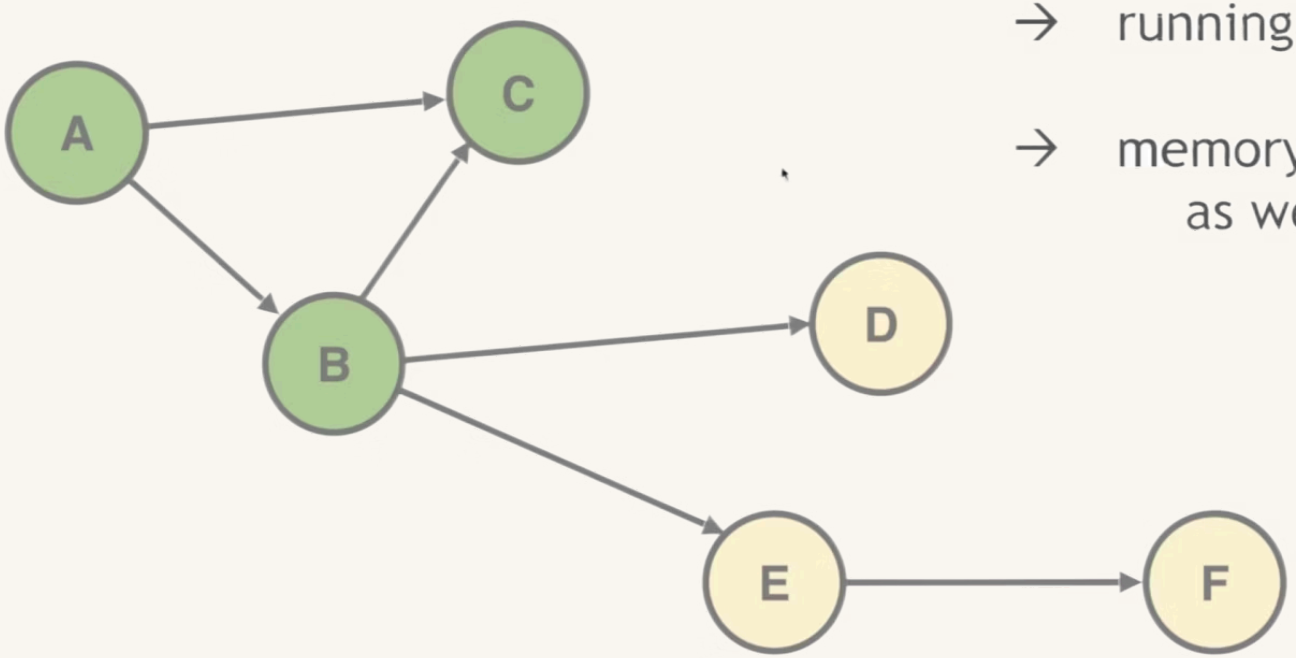
→ running time complexity is **O(V+E)**

→ memory complexity is not that favorable as we have to **store lot of references**

Breadth-First Search

There is a **G(V,E)** graph and the aim is to visit every **V** vertex exactly once

Breadth-first search visits the neighbors and then the neighbors of these new vertices until all nodes are visited



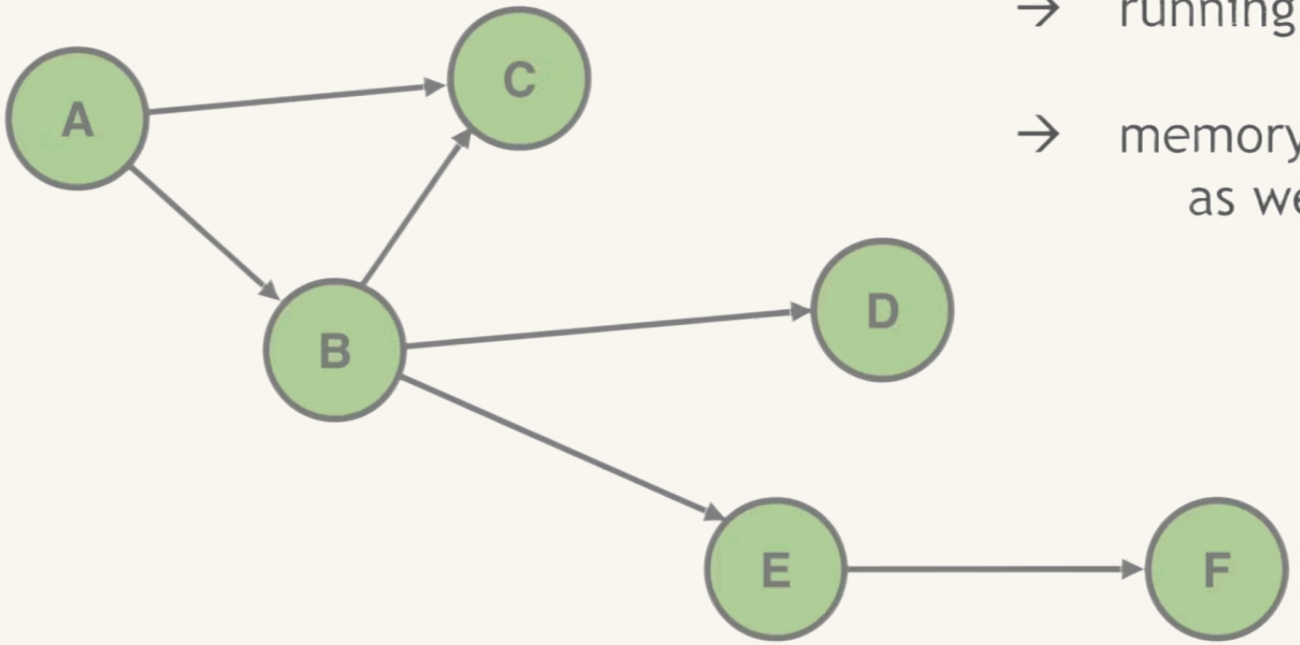
→ running time complexity is **O(V+E)**

→ memory complexity is not that favorable as we have to **store lot of references**

Breadth-First Search

There is a **G(V,E)** graph and the aim is to visit every **V** vertex exactly once

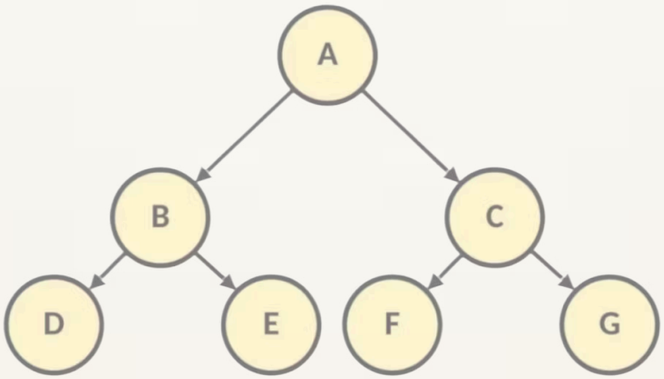
Breadth-first search visits the neighbors and then the neighbors of these new vertices until all nodes are visited



→ running time complexity is **O(V+E)**

→ memory complexity is not that favorable as we have to **store lot of references**

BFS Traversal Example 1



Start

Start from **Vertex A**

First Level

Visit neighbors: **B**, **C**

Second Level

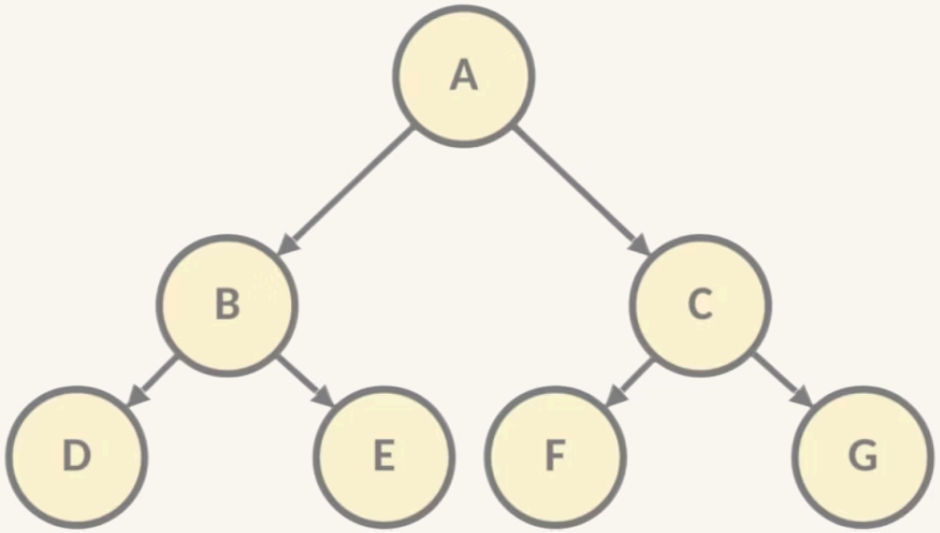
Then visit their neighbors: **D**, **E**, **F**, **G**

Key Rule

Key rule: Never revisit an already visited vertex

BFS traverses level by level

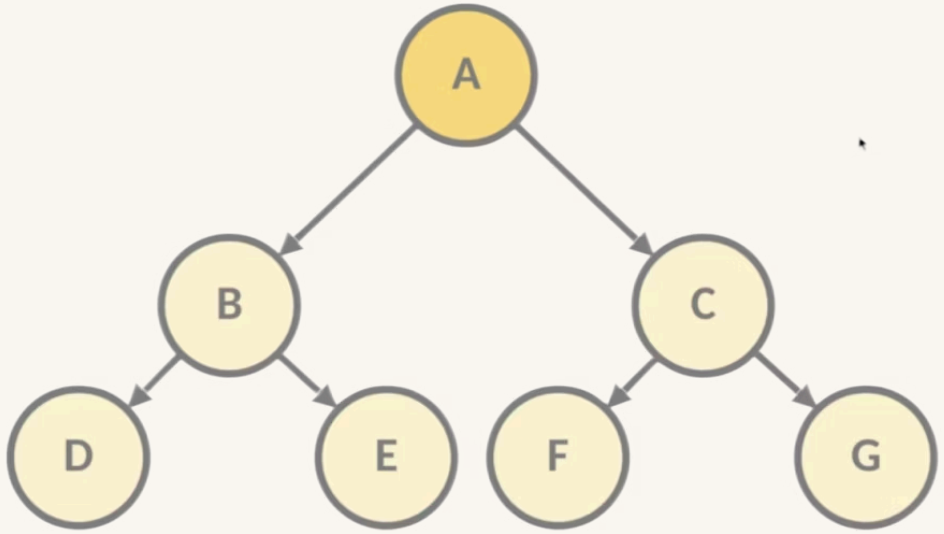
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

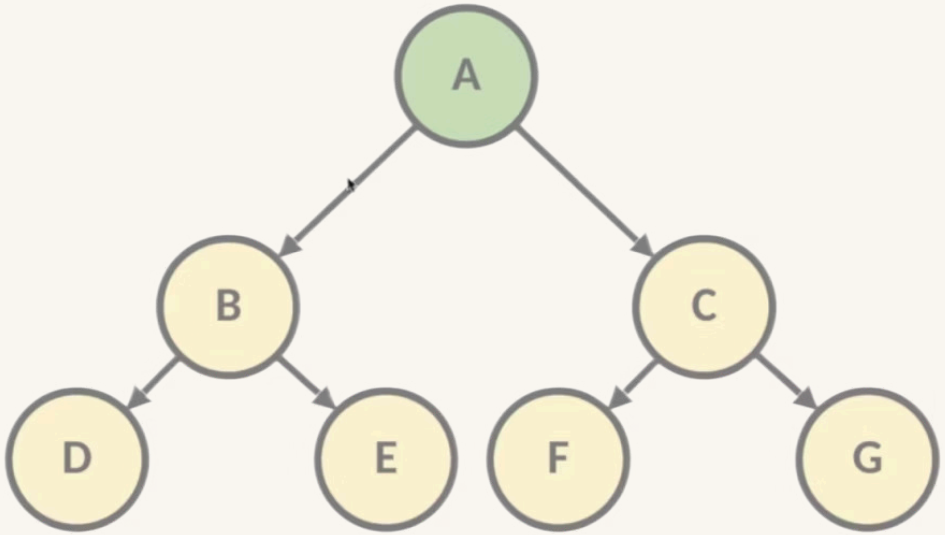
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

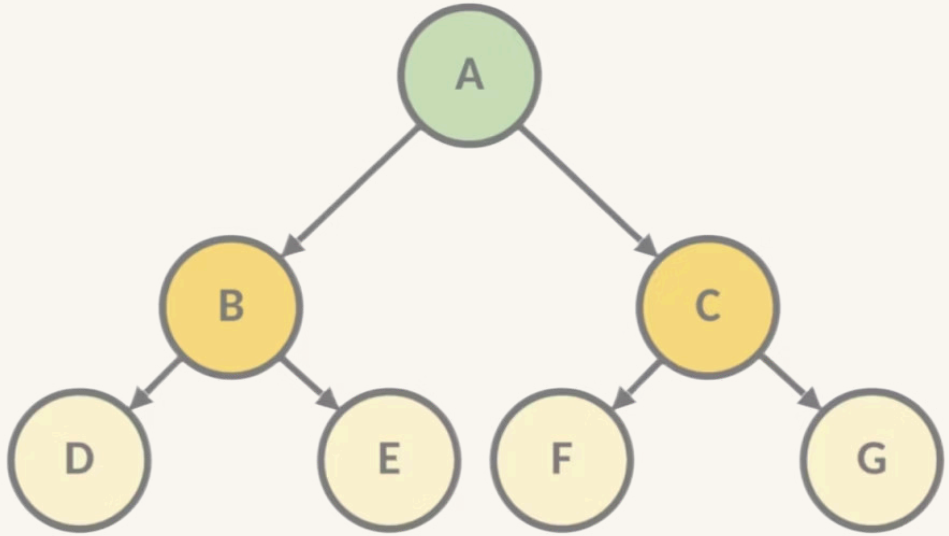
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

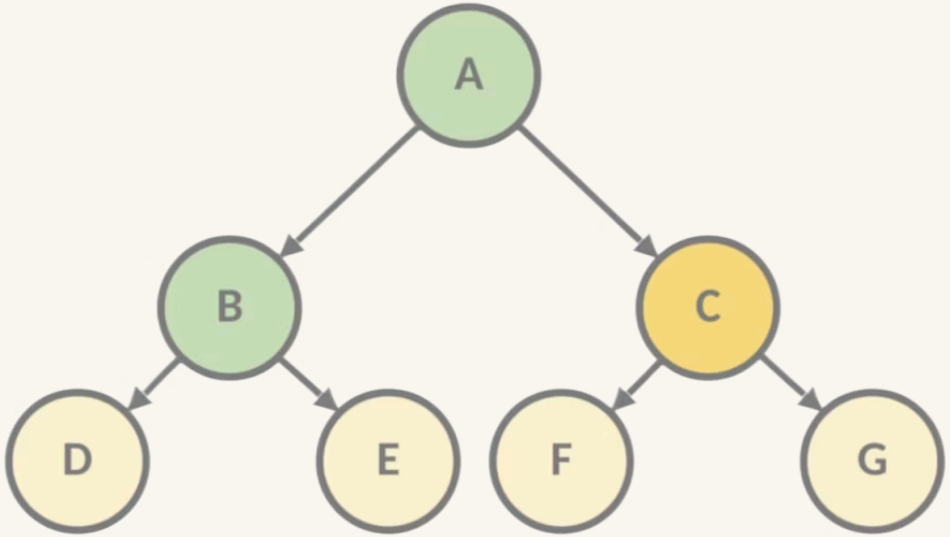
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

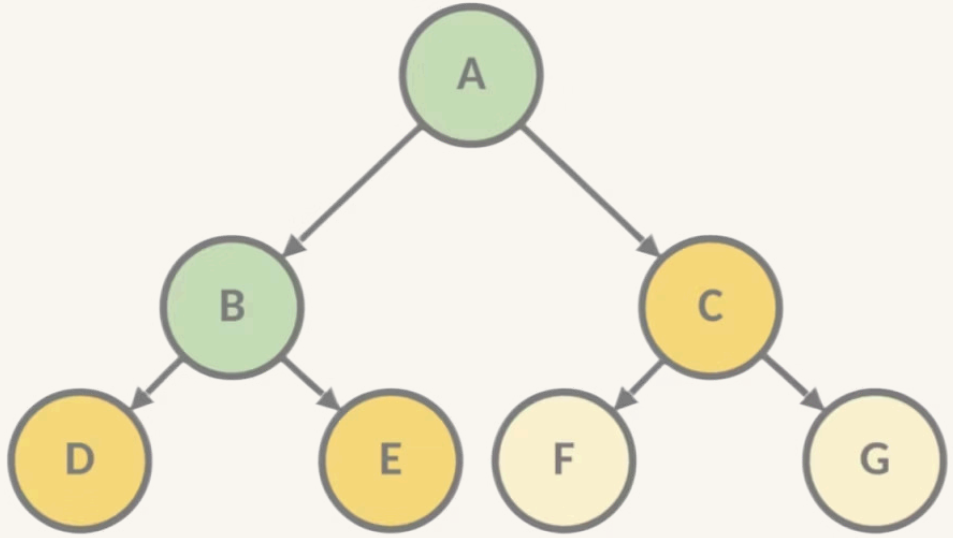
Breadth-First Search

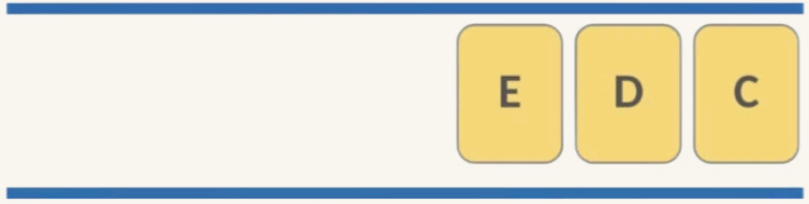


QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

Breadth-First Search

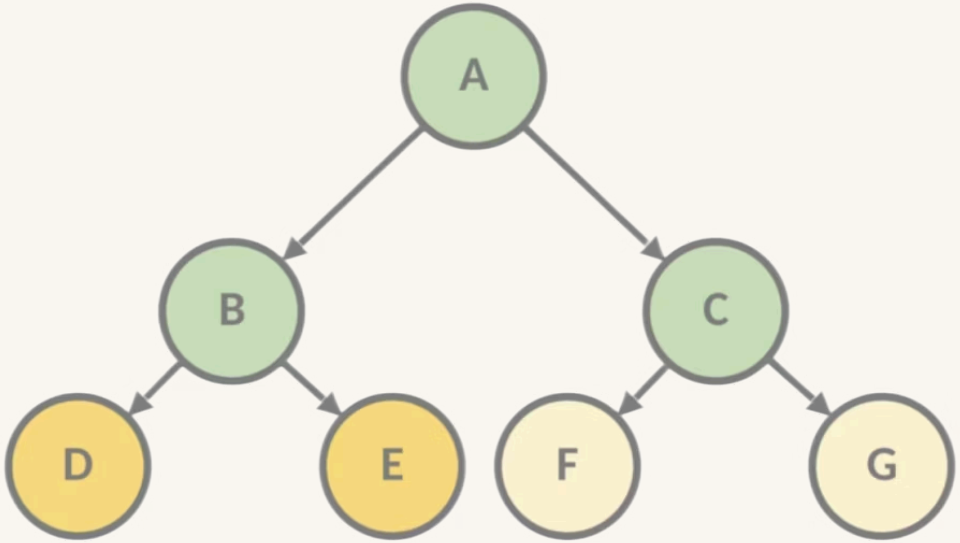




QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

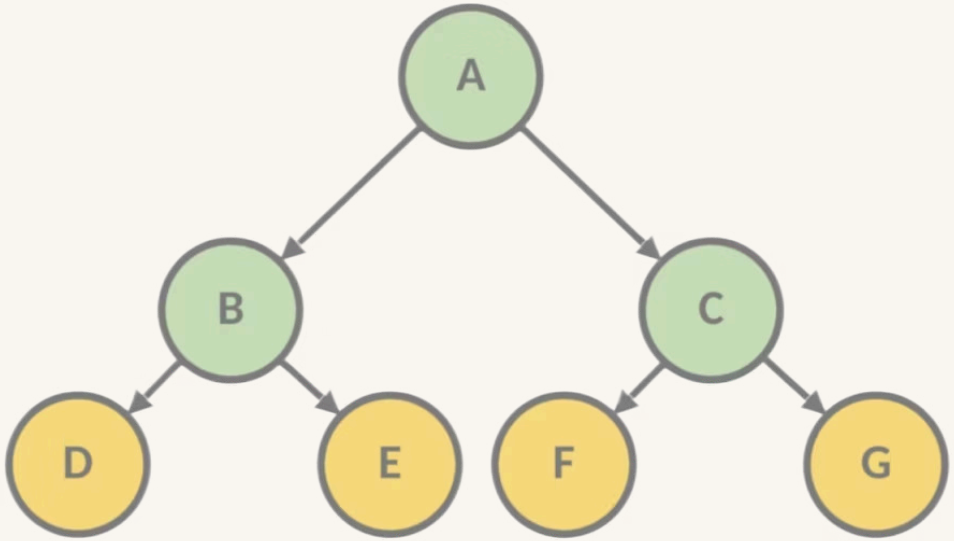
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

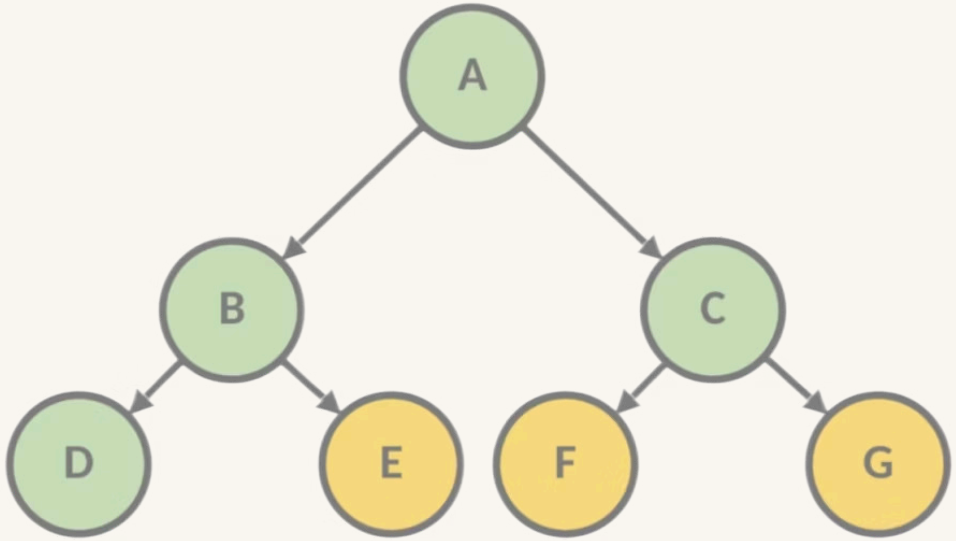
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

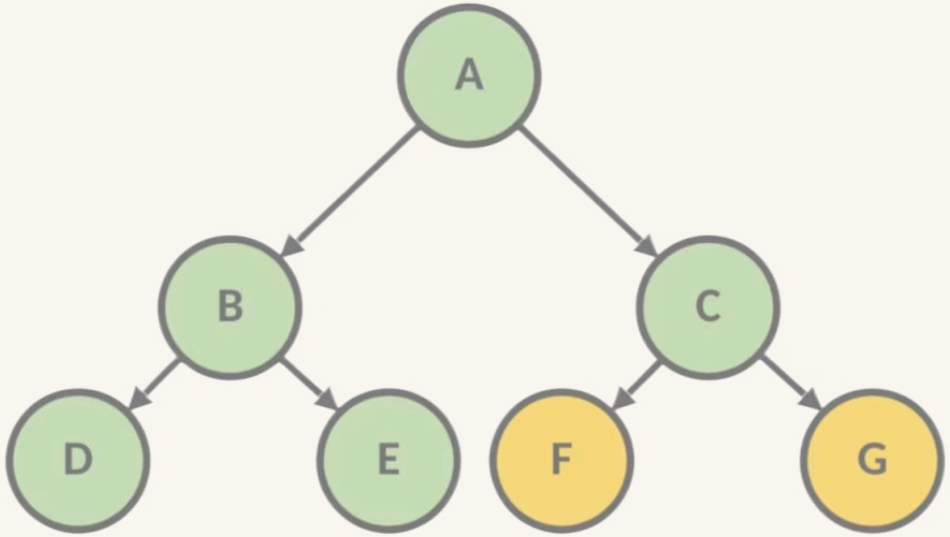
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

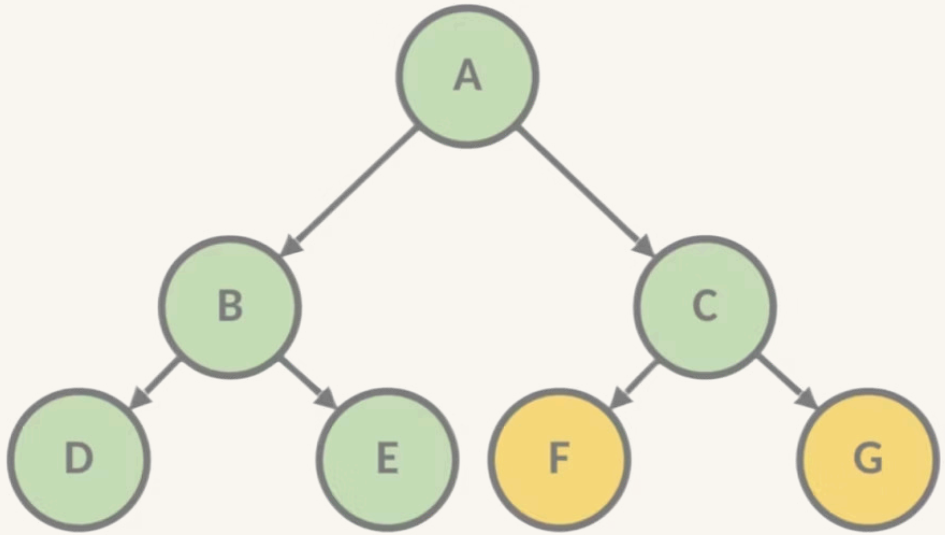
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

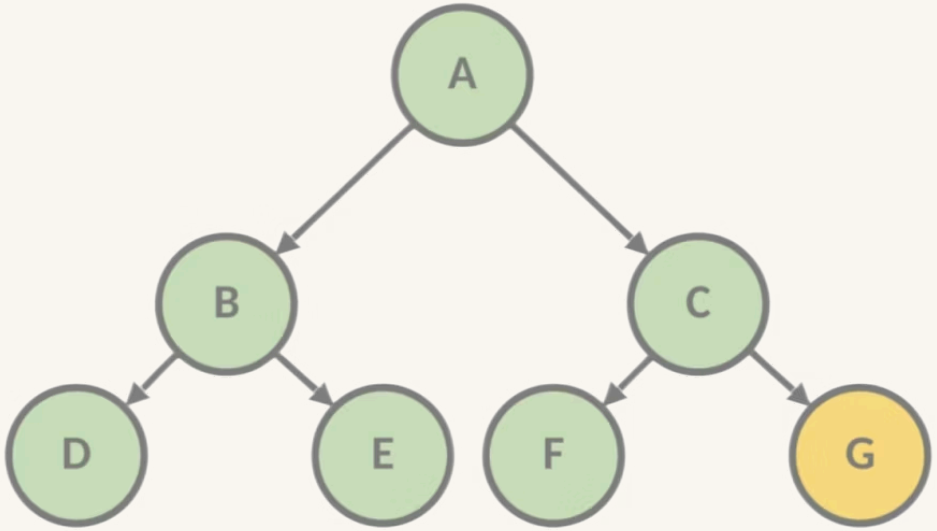
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

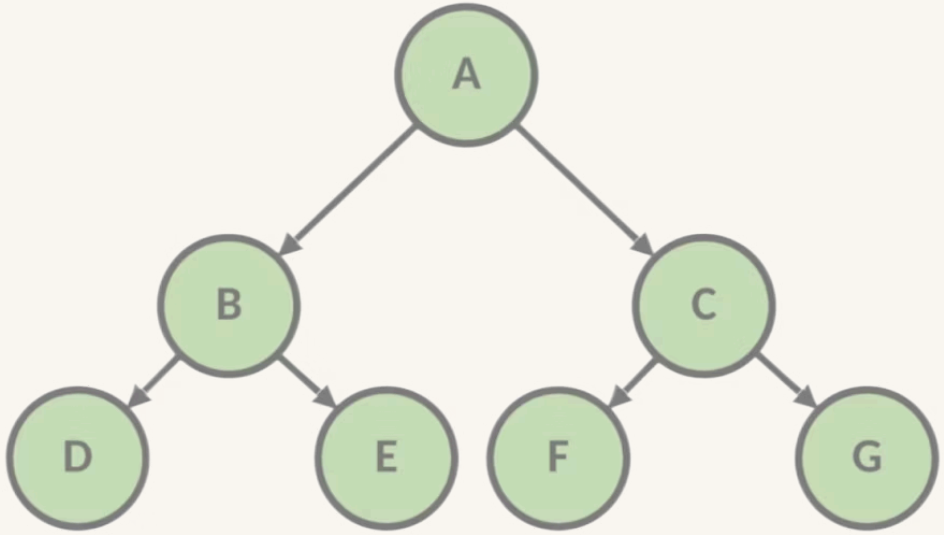
Breadth-First Search



QUEUE

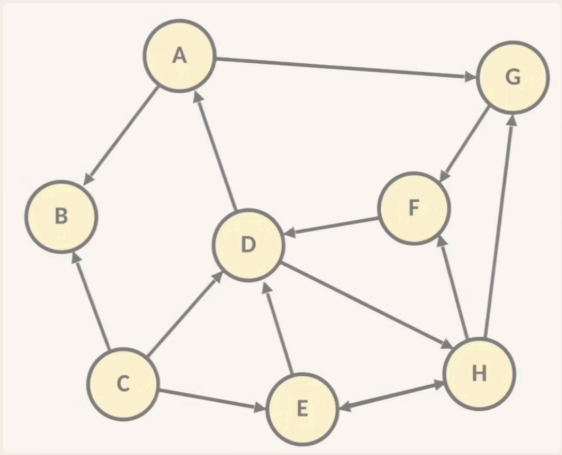
*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*



BFS Traversal Example 2

Initial State

Start at **Vertex C**

First Level

Enqueue neighbors: **B**, **D**, **E**

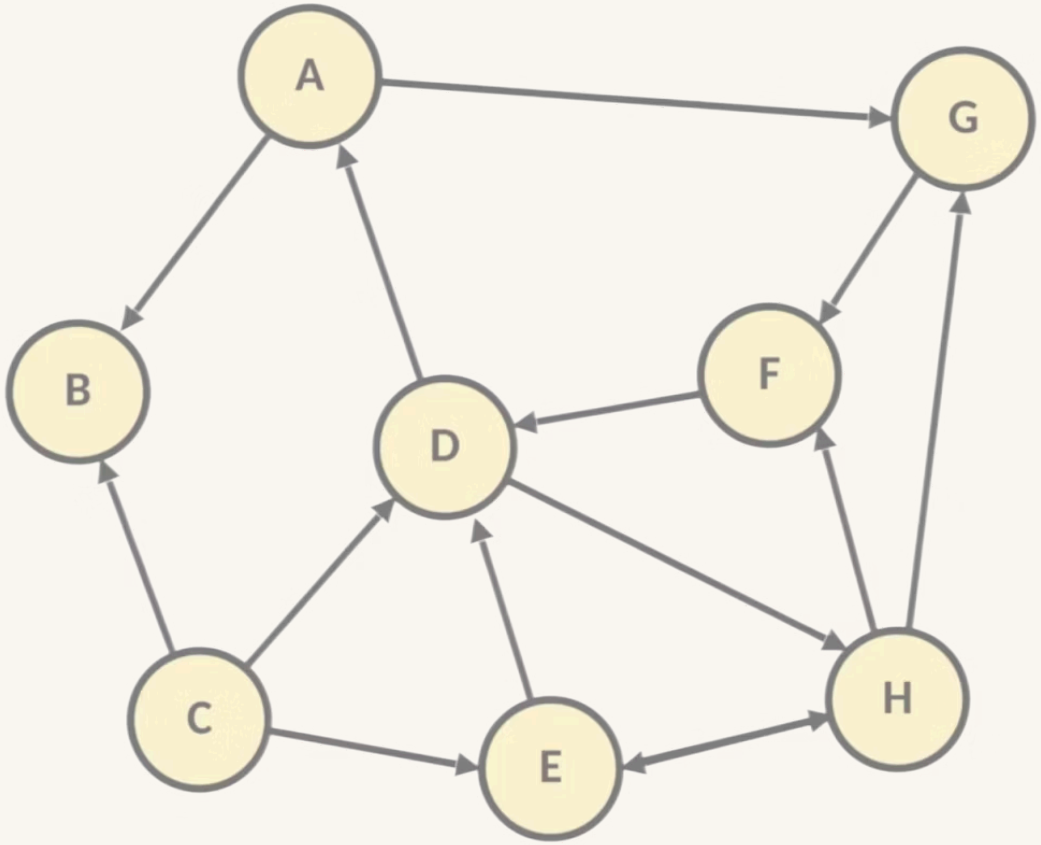
Continue Exploration

Visit **H**, then **A**, **F**, **G**

Avoid Duplicates

Skip vertices already visited or enqueued

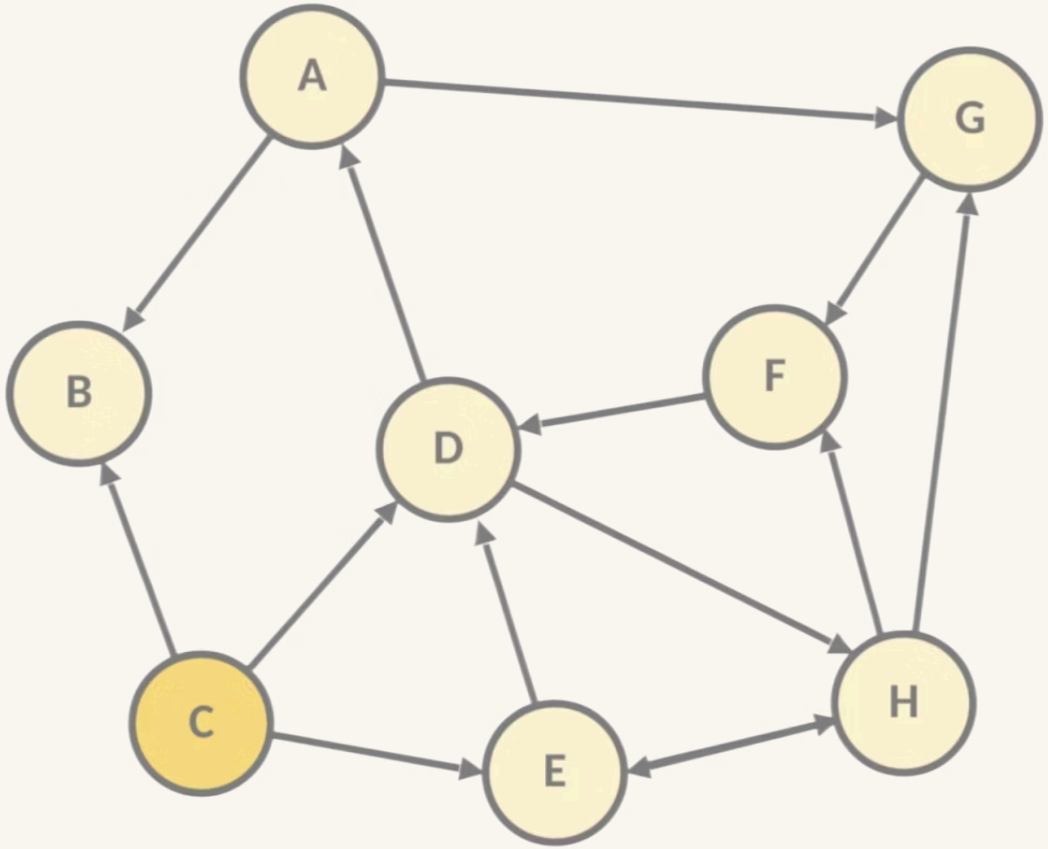
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

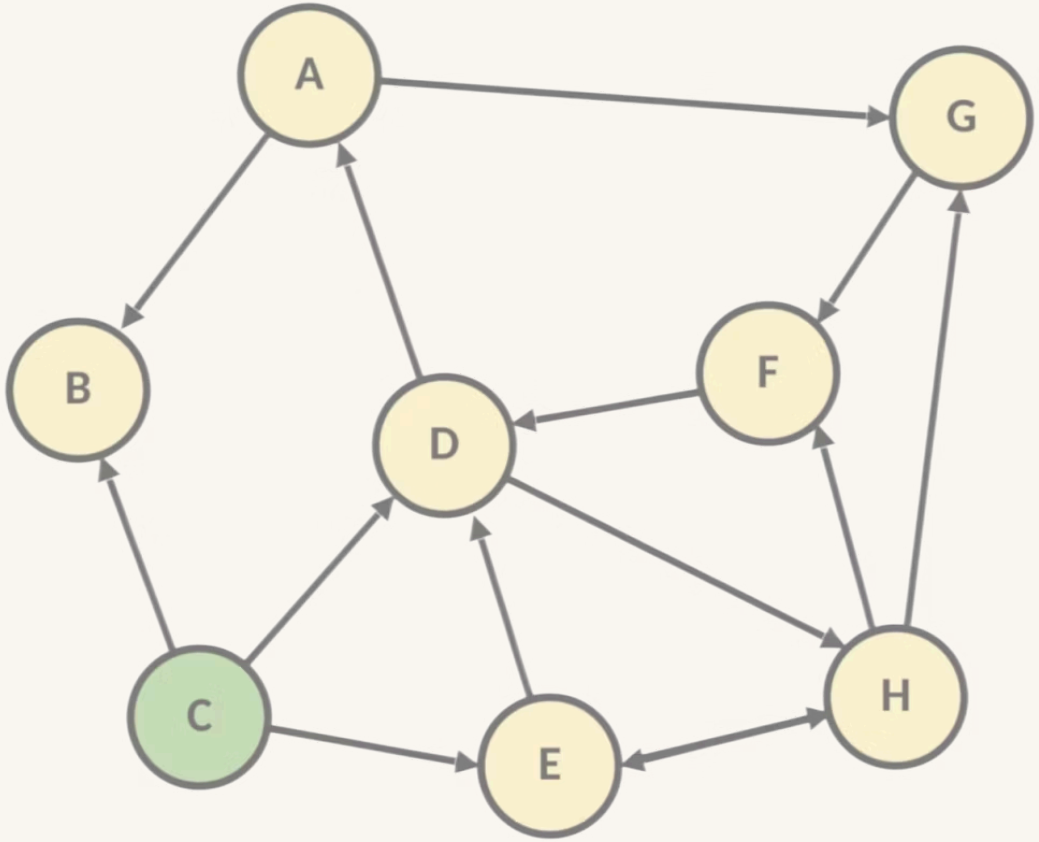
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

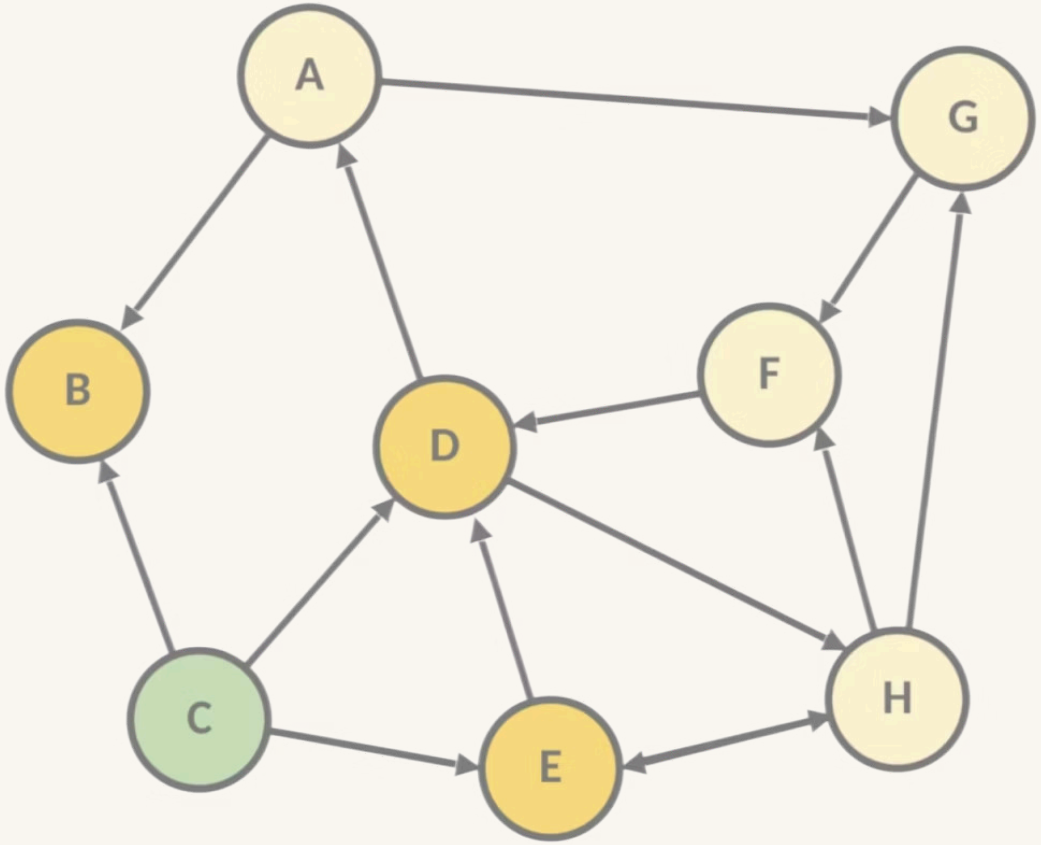
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

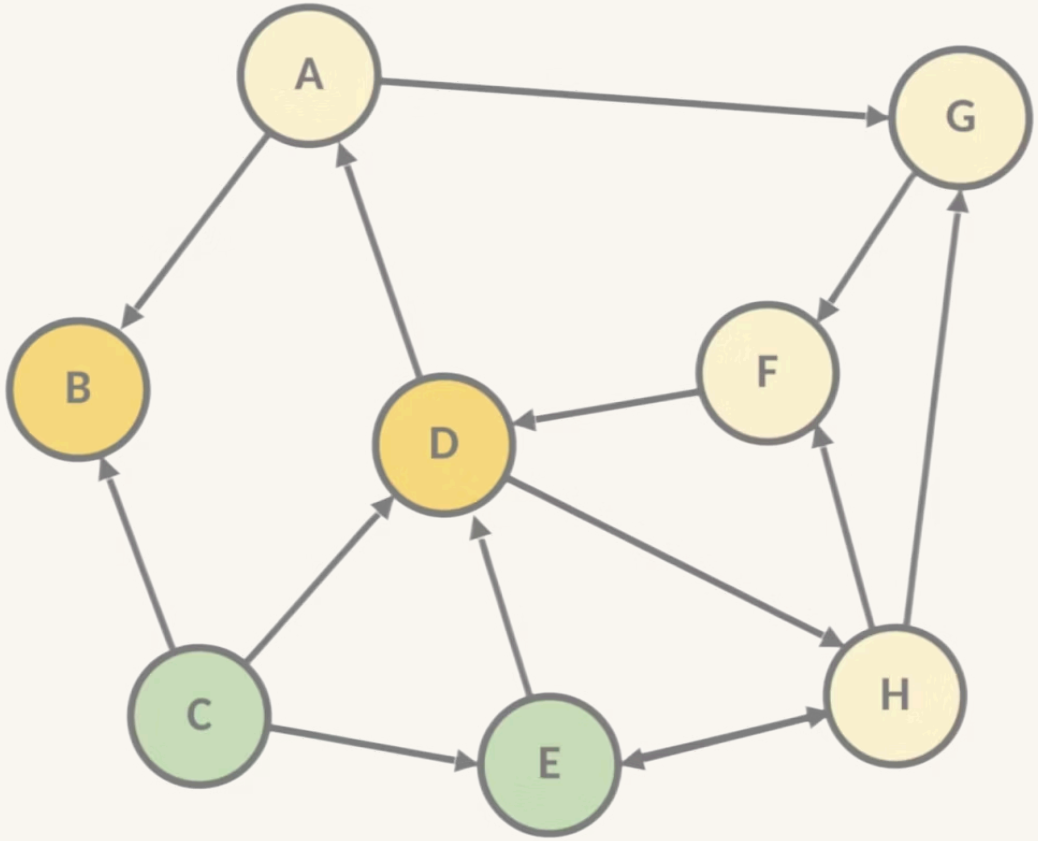
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

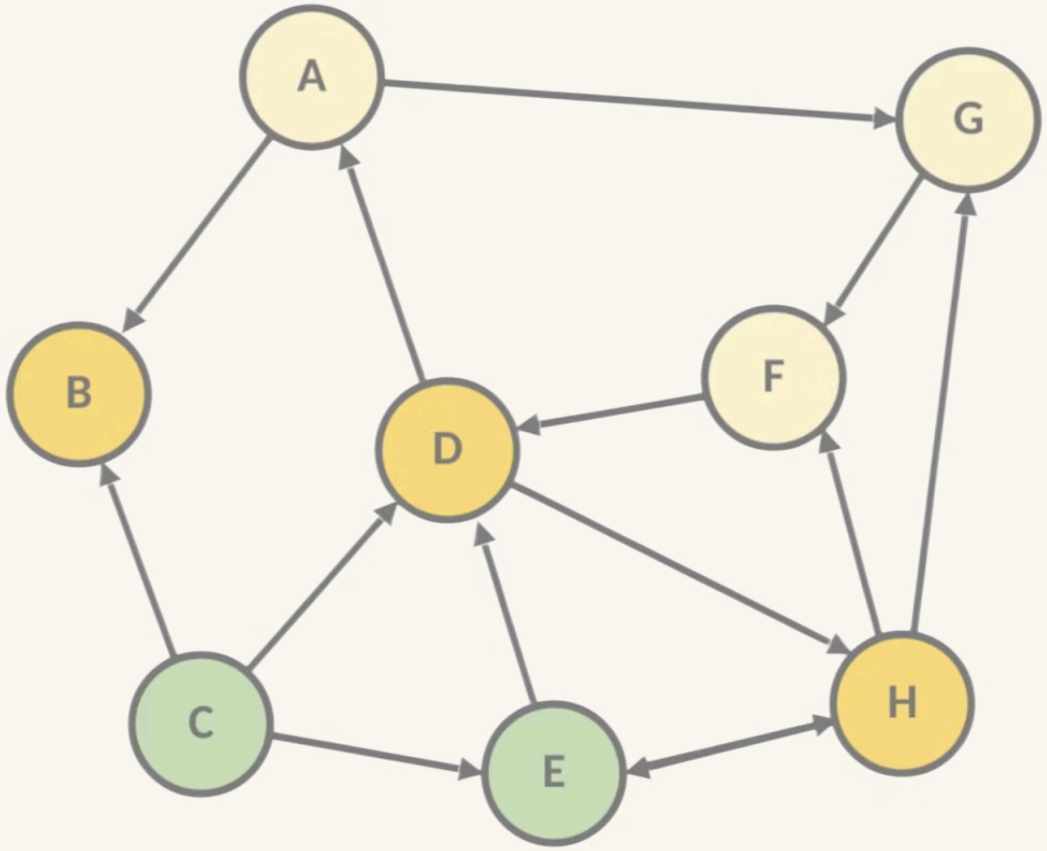
Breadth-First Search

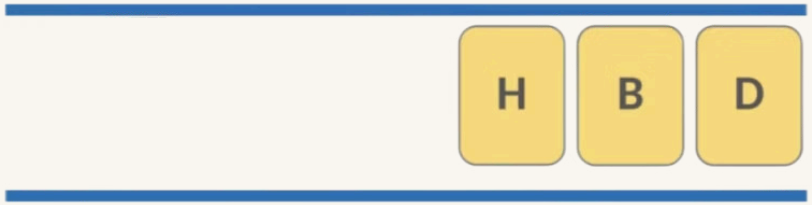


QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

Breadth-First Search

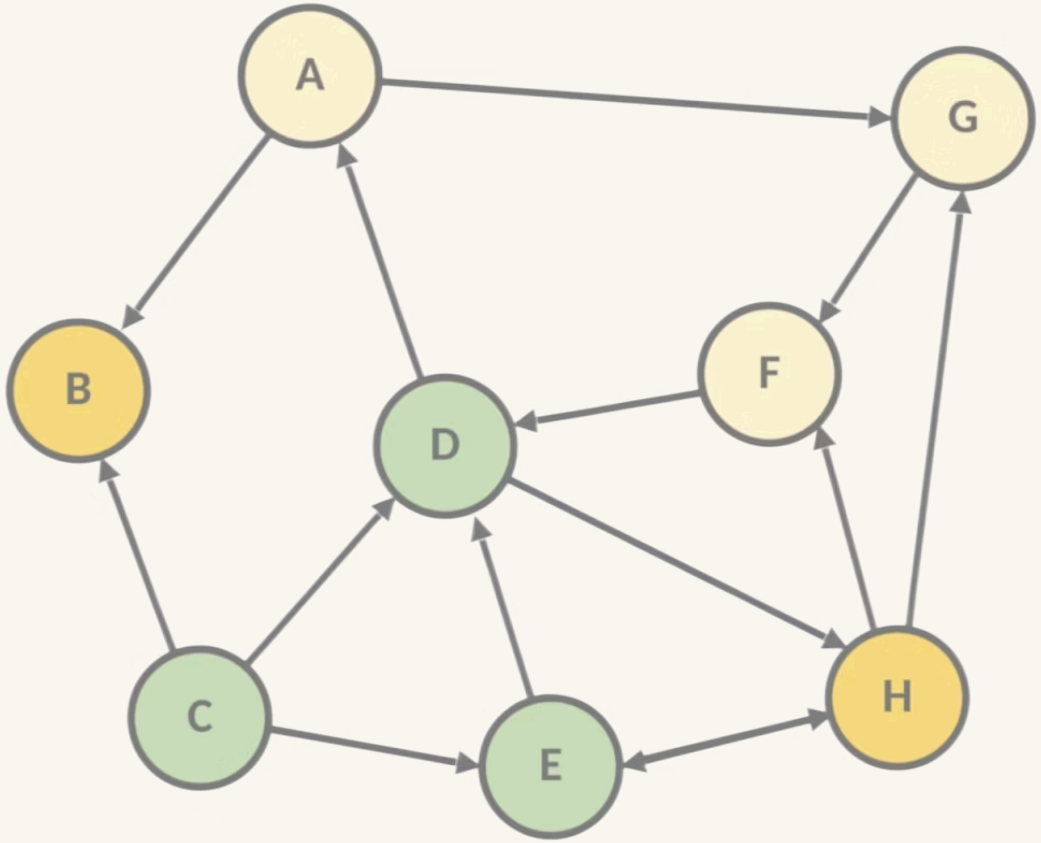




QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

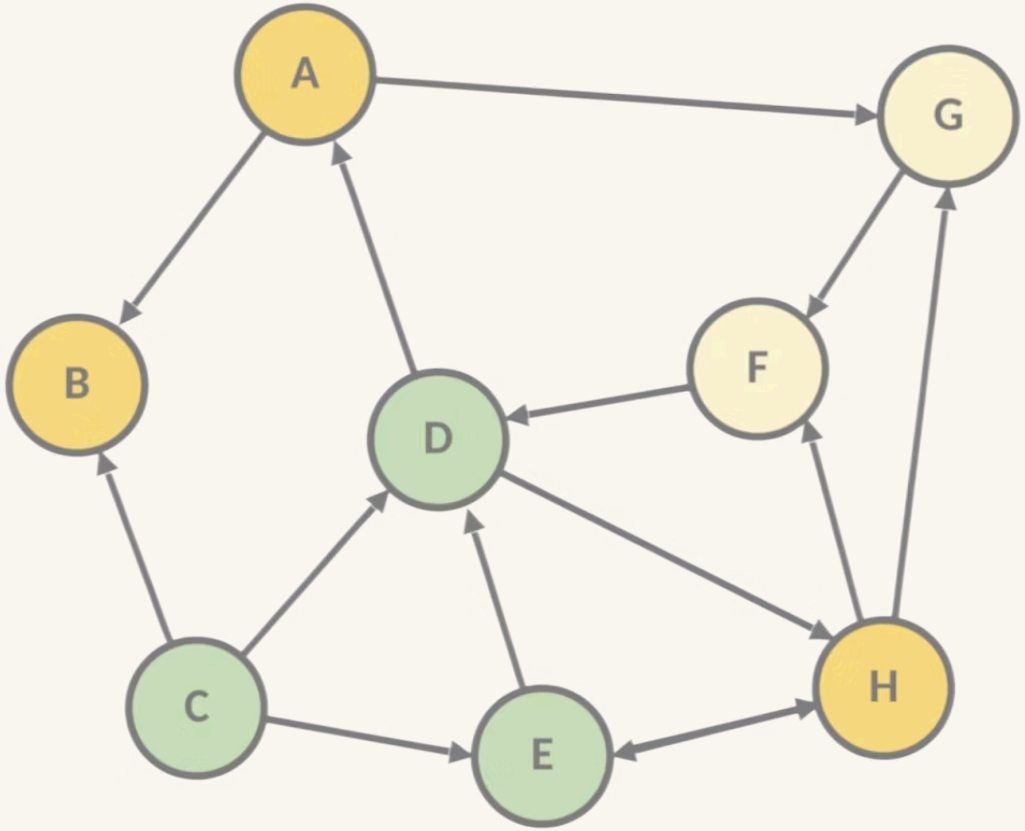
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

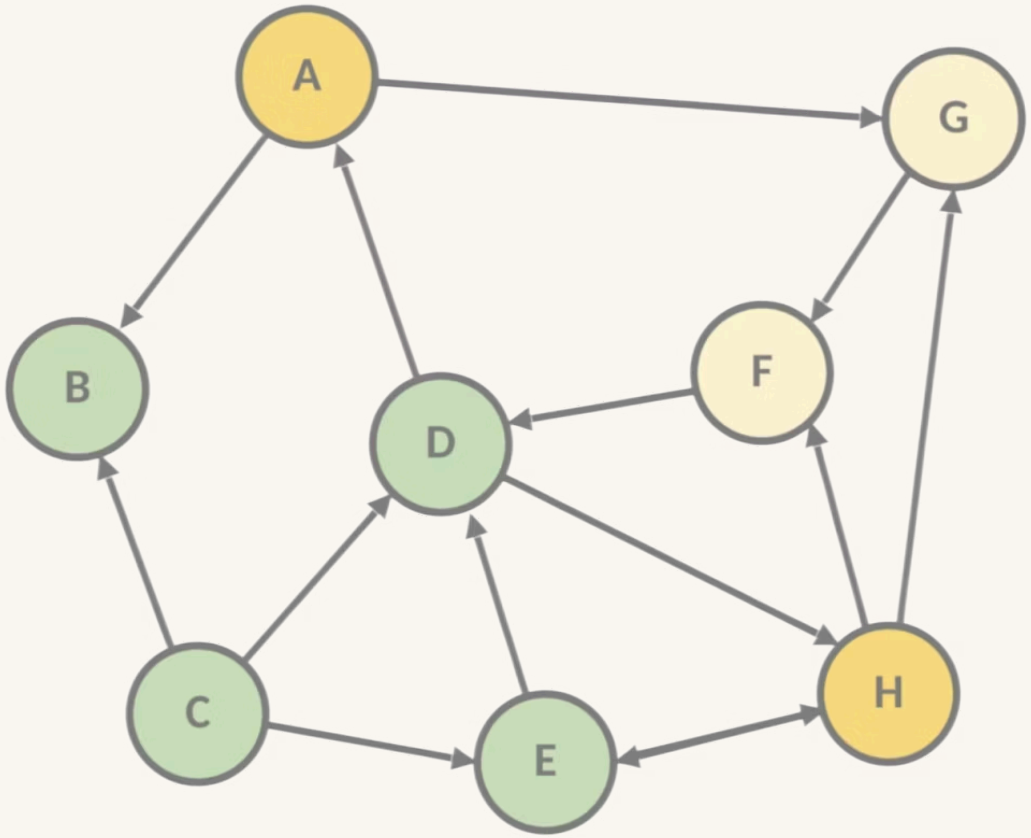
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

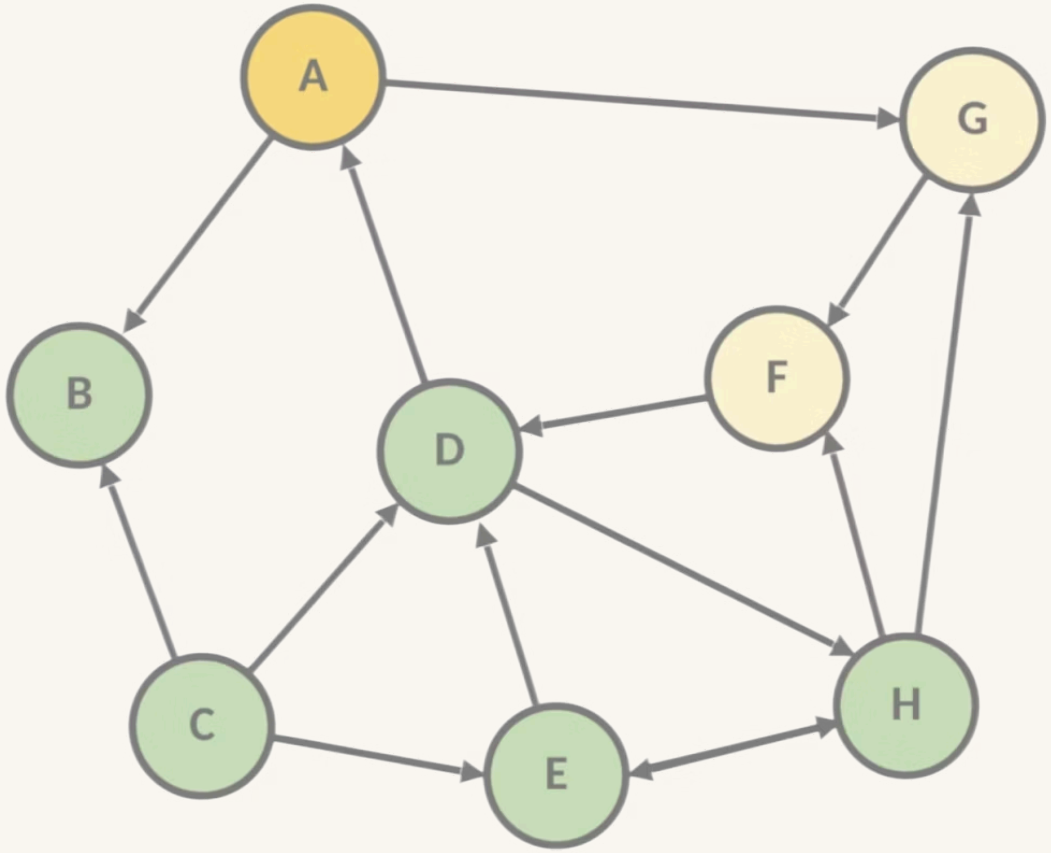
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

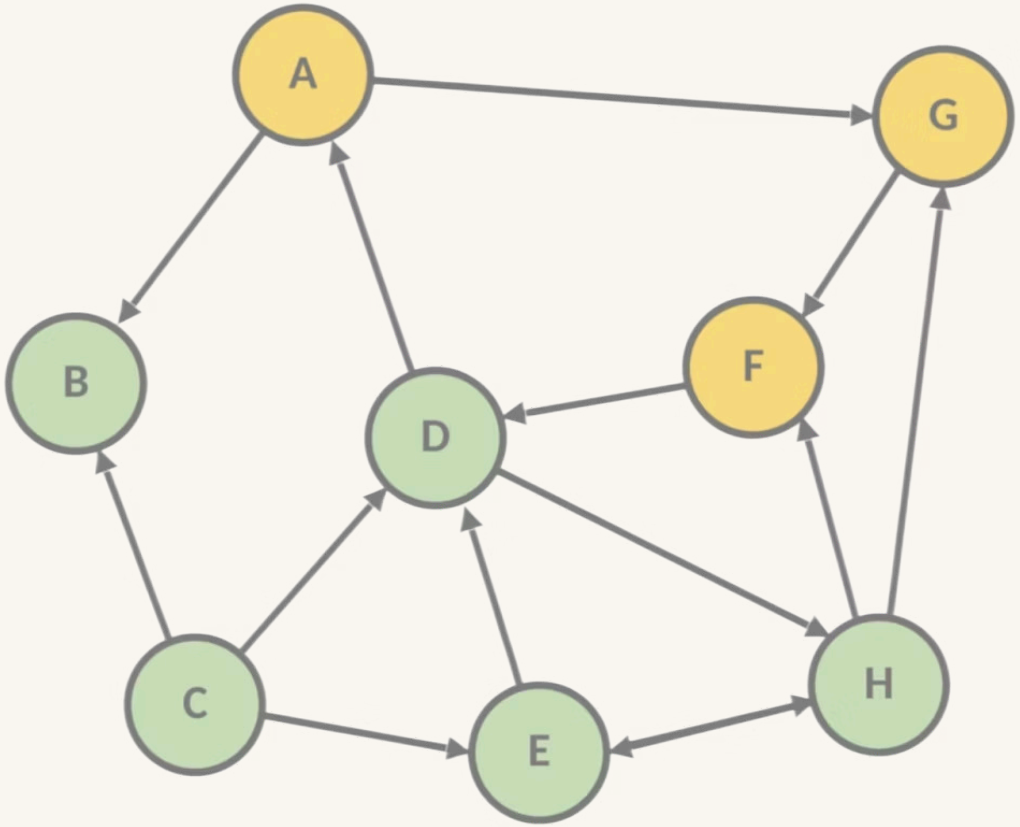
Breadth-First Search

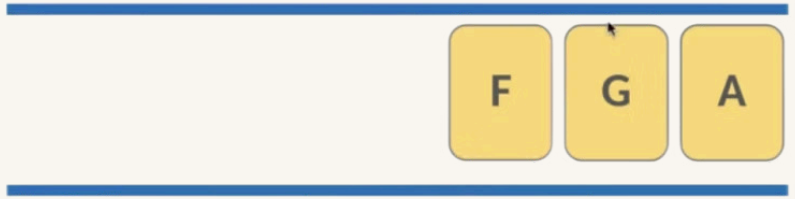


QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

Breadth-First Search

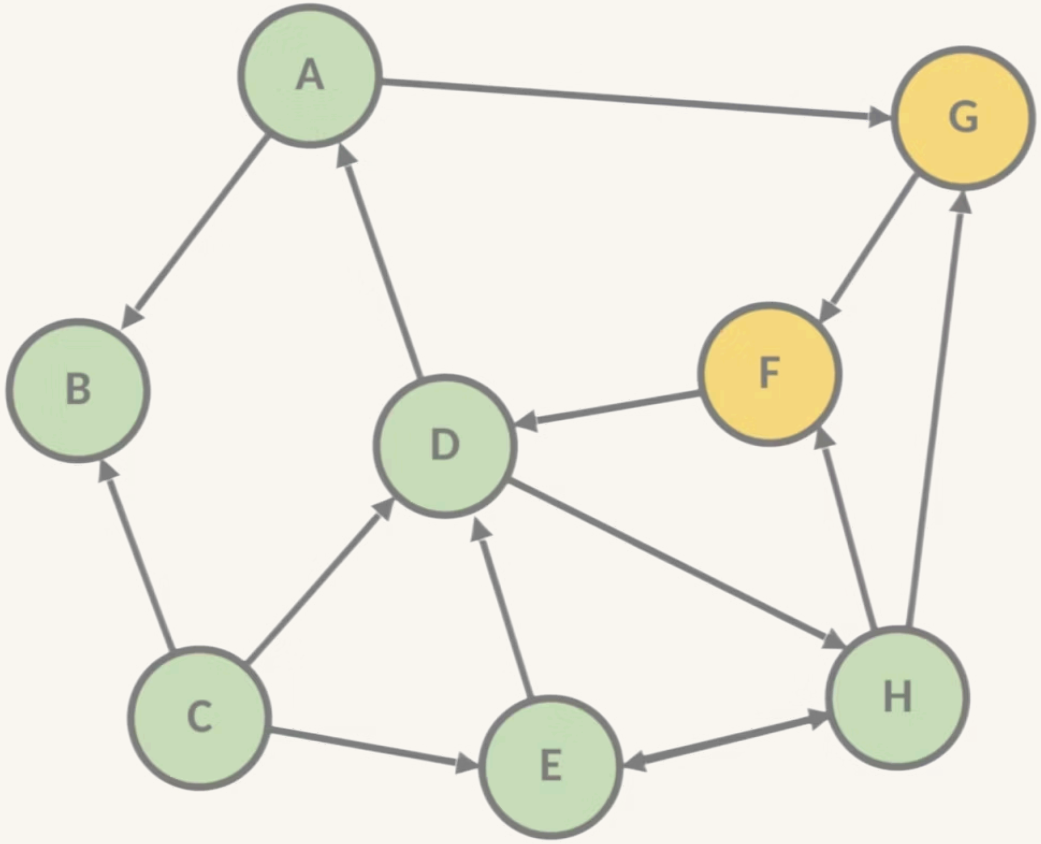




QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

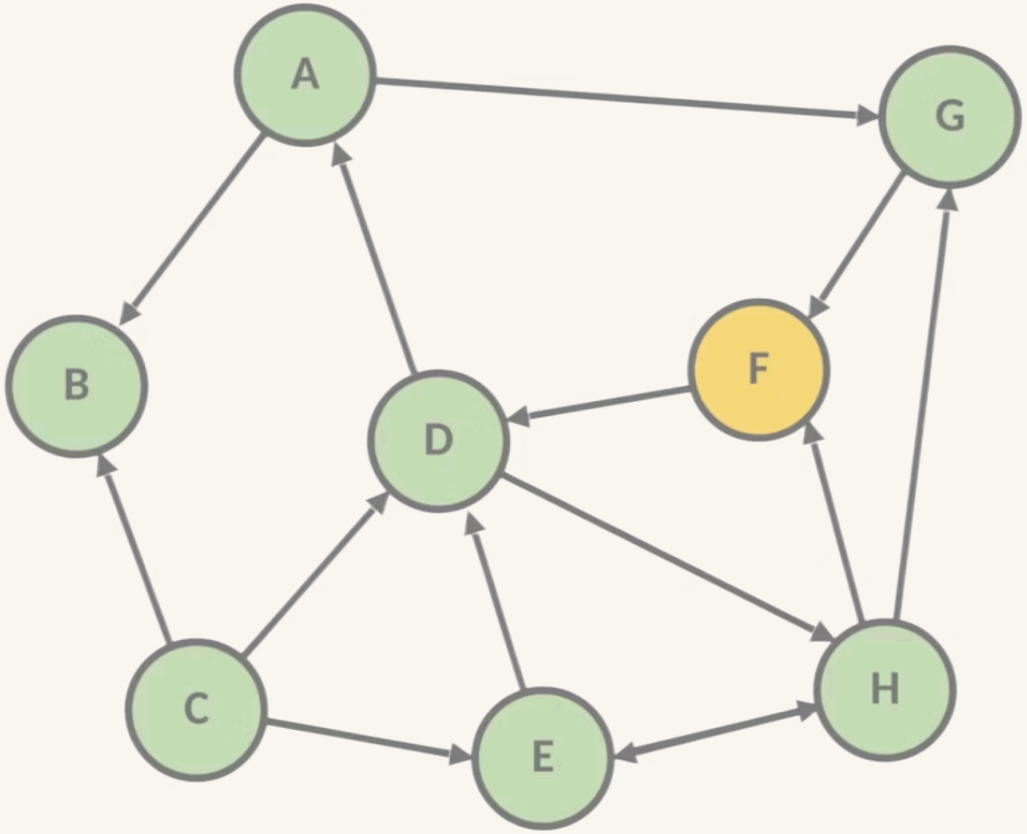
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

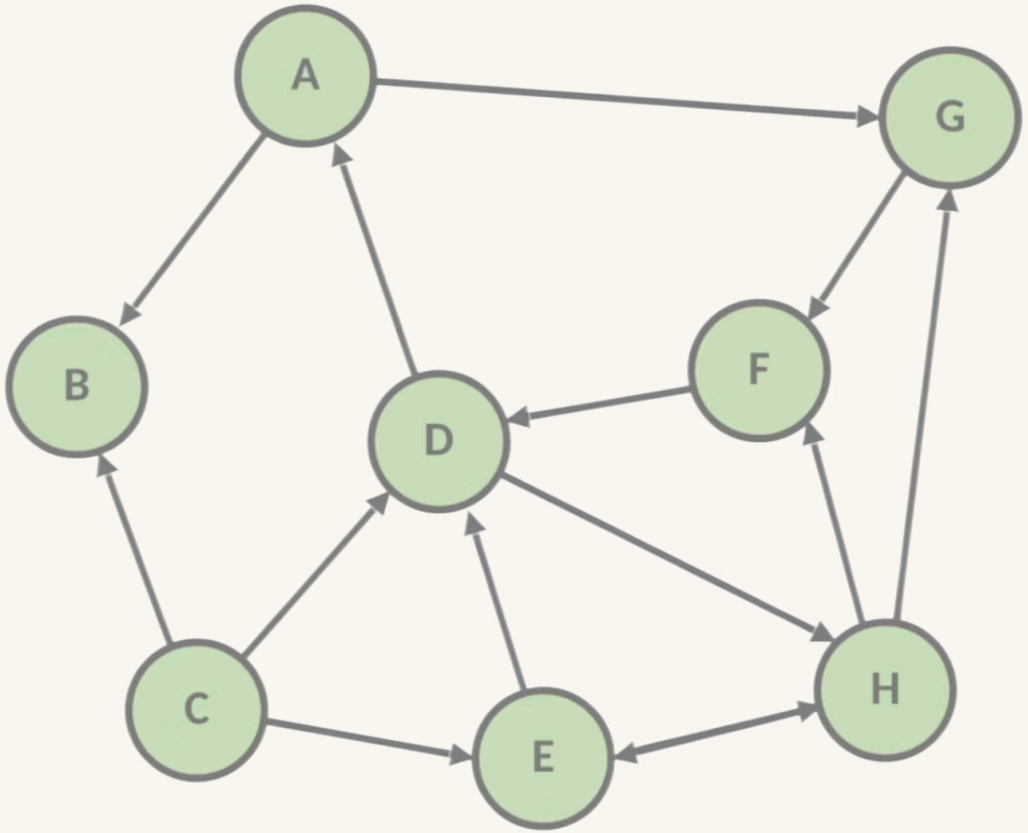
Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

Breadth-First Search



QUEUE

*storing the **references** (objects)*
*in a **FIFO** first-in-first-out order*

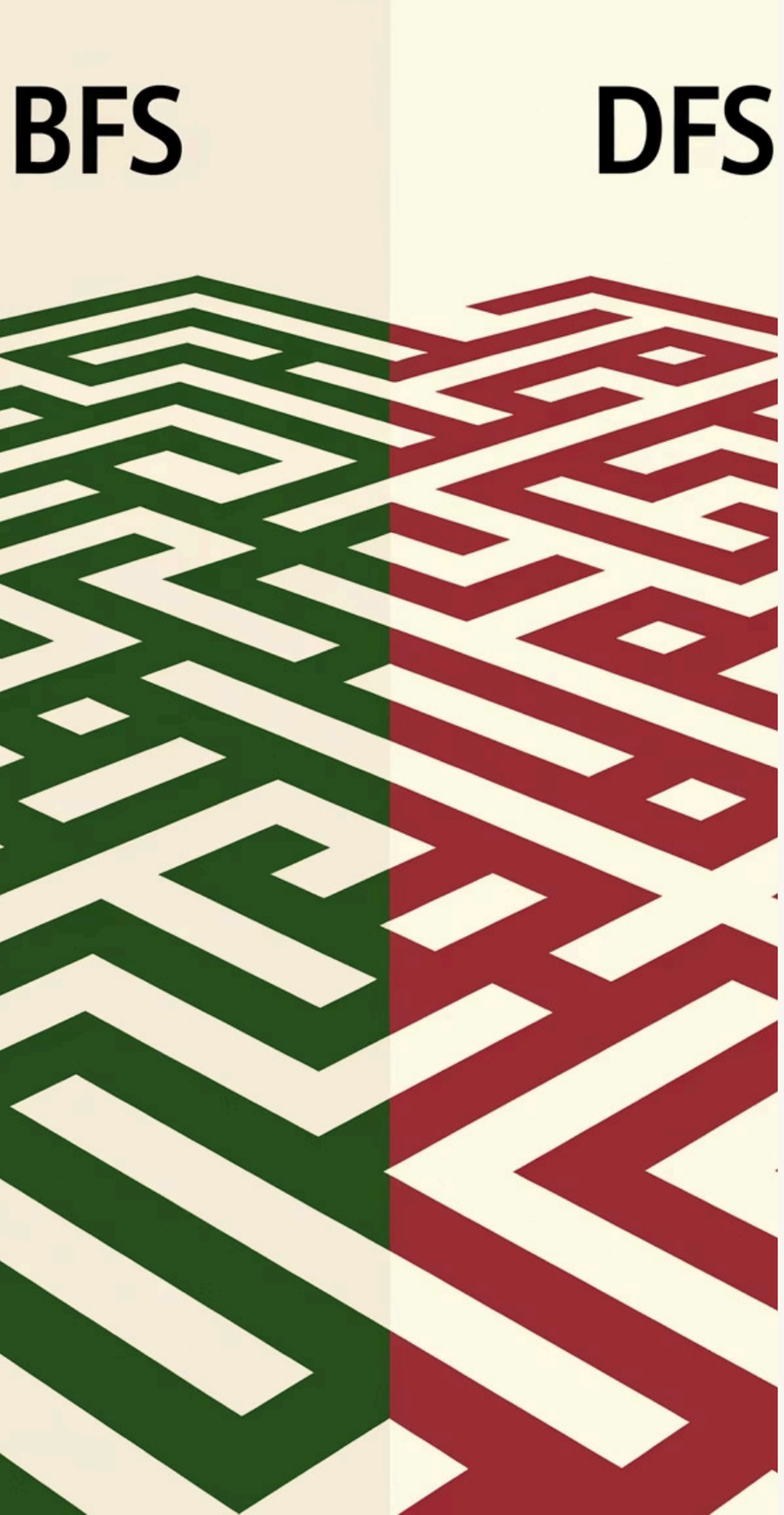
BFS – Time and Memory Complexity

Time Complexity: O(V + E)

- V: number of vertices
- E: number of edges

Memory Complexity: Higher than DFS

- BFS needs to store multiple references at once
- Requires a **queue** (FIFO structure)



BFS Builds Shortest Path

Shortest Path Guarantee

BFS constructs the shortest path when all edge weights = 1

Similar to Dijkstra

Dijkstra's algorithm works similarly under this assumption

DFS Limitation

DFS does **not** guarantee shortest path

BFS Advantage

This is a key **advantage of BFS** over **DFS**

Breadth-First Search

```
bfs(vertex):

        Queue queue
        set vertex to be visited
        queue.add(vertex)

        while queue not empty

            actual = queue.dequeue()

            for v in actual neighbors
                if v is not visited
                    set v visited
                    queue.add(v)
```

→ the underlying abstract data type is a **queue (FIFO)**

→ we have an empty queue at the beginning and we keep checking whether we have visited the nodes or not

→ the algorithm terminates when there are no more nodes in the **queue** abstract data type

Pseudocode Overview

Initialize

Mark the **starting vertex** as visited

Add it to the queue

Expand

Mark them visited and enqueue them

Repeat until queue is empty

Main Loop

While the queue is not empty

Process

Dequeue a vertex

Visit all unvisited neighbors

BFS Implementation Strategy

Initialize Queue

Use a **Queue** (FIFO – First In, First Out)

Start by inserting the root node into the queue

Process Queue

While queue is not empty:

Visit Nodes

Dequeue the first item

Visit and enqueue its unvisited neighbors

FIFO Queue Behavior

Queue Data Structure

BFS uses a **queue** to control traversal order

FIFO Principle

FIFO = First In, First Out

Ensures layer-by-layer exploration

Processing Order

Vertices are dequeued and visited in insertion order

Summary of BFS

Level-Based Traversal

BFS visits vertices **by levels**

Shortest Path Guarantee

Ensures **shortest path** in **unweighted graphs**

Memory Intensive

Requires **more memory** than **DFS**

Ideal Applications

Navigation problems, Pathfinding in games, State-space exploration in AI