

# Depth First Search & Breadth First Search

---

**Md. Maskaowat Ahsan Hasby**

Student ID: 2410876132

Dept. of Computer Science and Engineering

University of Rajshahi

# Contents:

- ▶ Introduction of Graphs
- ▶ Linked Representation of a Graph
- ▶ Breadth First Search (BFS)
- ▶ Depth First Search (DFS)
- ▶ Differences Between BFS and DFS

# Introduction of Graphs

---

## Definition

### ► Graphs:

A Graph **G** is a non-linear data structure defined by an ordered pair (**V**, **E**) where:

- ❖ **V (Vertices)**: A set of nodes representing data entities.
- ❖ **E (Edges)**: A set of connections between pairs of vertices.

Graphs represent pairwise relationships between objects (e.g., cities connected by roads).

## Memory Representations

### 1. Sequential Representation

- Uses an **Adjacency Matrix** (2D Array). Good for dense graphs.

### 2. Linked Representation

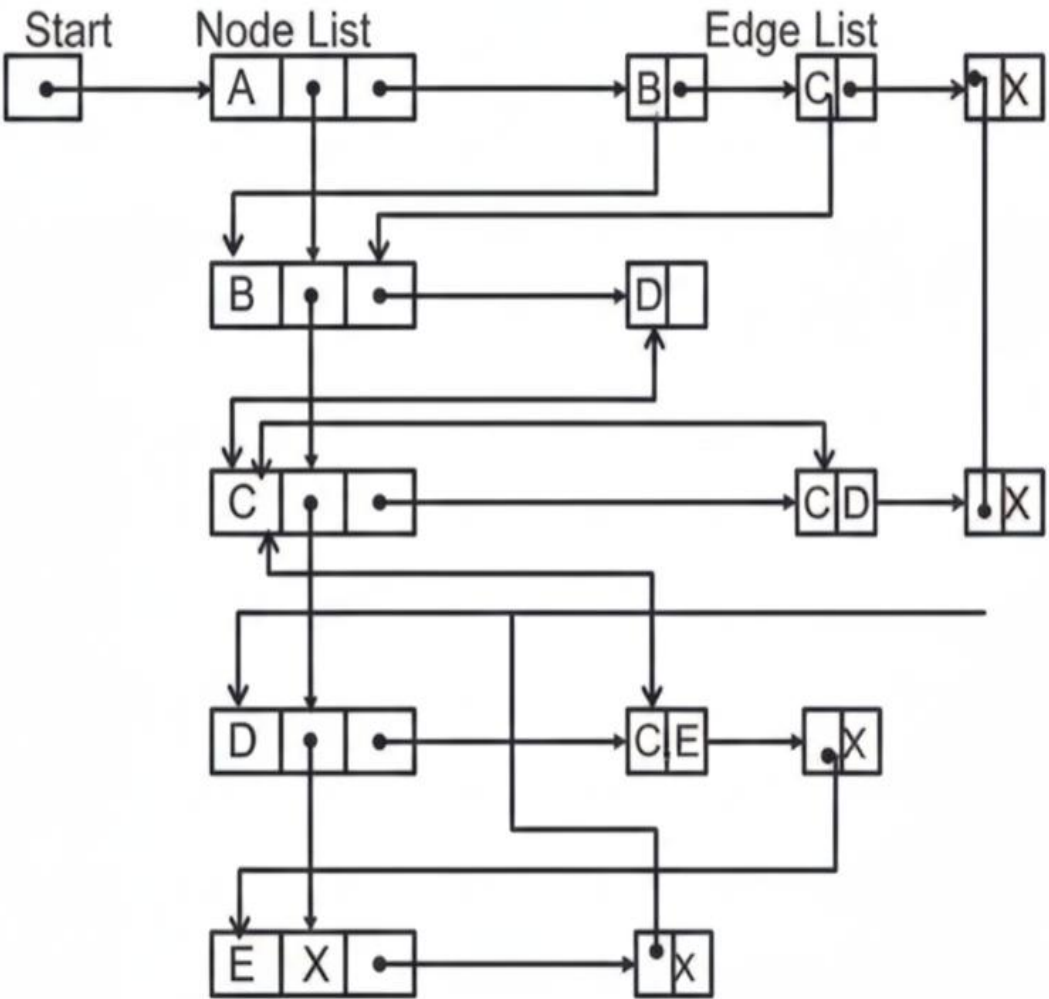
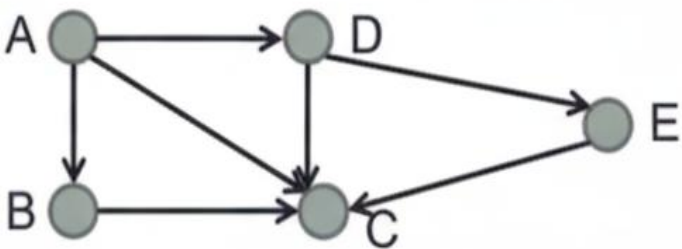
- Uses **Adjacency Lists** (Linked Lists). Good for sparse graphs.

# Linked Representation Overview

## The Concept

- ❑ In linked representation, each node in G is followed by its **adjacency list**.
- ❑ This list contains all its "successors" or "neighbors".
- ❑ This representation uses two specific lists:
  1. Node list (NODE)
  2. Edge list (EDGE)

Node	Adjacency List
A	B, C, D
B	C
C	
D	X, E
E	C



# Graph Traversal & Node States

---

There are two ways:

1. **Breadth First Search (BFS)**: Uses a queue as an auxiliary structure to hold nodes for future processing.
2. **Depth First Search (DFS)**: Uses a stack as an auxiliary structure.

❖ During algorithms (BFS/DFS), each node N will be in one of three states:

1

## Ready State

The initial state of Node N.  
(STATUS=1)

2

## Waiting State

Node N is in the Queue or Stack,  
waiting to be processed.  
(STATUS=2)

3

## Processed State

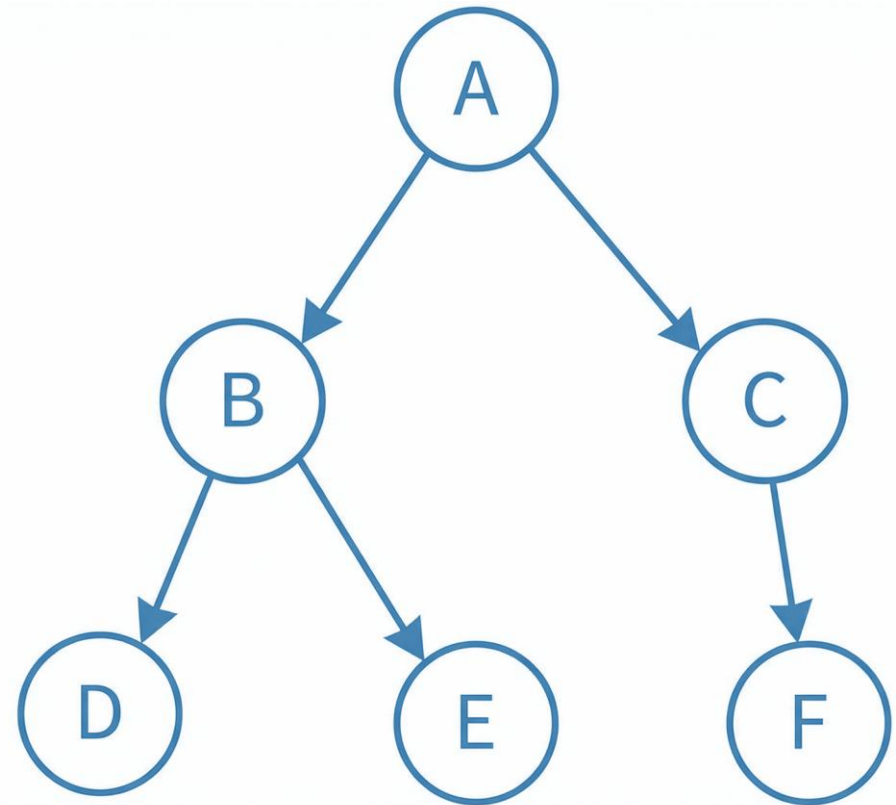
Node N has been fully processed.  
(STATUS=3)

# Breadth-First Search (BFS)

---

## ❖ Algorithm Logic:

1. Initialize all nodes to **Ready (1)**
2. Put start node A in **Queue** and set to **Waiting (2)**.
3. Repeat until Queue is empty:
  - (i) Remove front node N. Process it. Set to **Processed (3)**.
  - (ii) Add all neighbors of N that are **Ready (1)** to Queue and set them to **Waiting (2)**.



**Breadth-First Search**

# BFS Simulation Steps:

## Initialization

- **Queue** : Empty initially.
- **Visited Nodes** : Start with only the source node marked as visited.

## **Step 1 : Enqueue the starting node**

- Enqueue node **A**.
- **Queue** : [A]
- **Visited** : {A}

## **Step 2: Dequeue and process node A**

- Dequeue **A** and process it.
- Enqueue its unvisited neighbors : **B** and **C**.
- **Queue** : [B, C]
- **Visited** : {A, B, C}

## **Step 3 : Dequeue and process node B**

- Dequeue **B** and process it.
- Enqueue its unvisited neighbors: **D** and **E**.
- **Queue** : [C, D, E]
- **Visited** : {A, B, C, D, E}

## **Step 4 : Dequeue and process node C**

- Dequeue **C** and process it.
- Enqueue its unvisited neighbor : **F**.
- **Queue** : [D, E, F]
- **Visited** : {A, B, C, D, E, F}

## **Step 5 : Process the remaining nodes**

- Continue dequeuing and processing nodes (**D, E, F**) until the queue is empty.

**Final Traversal Order :  $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$**

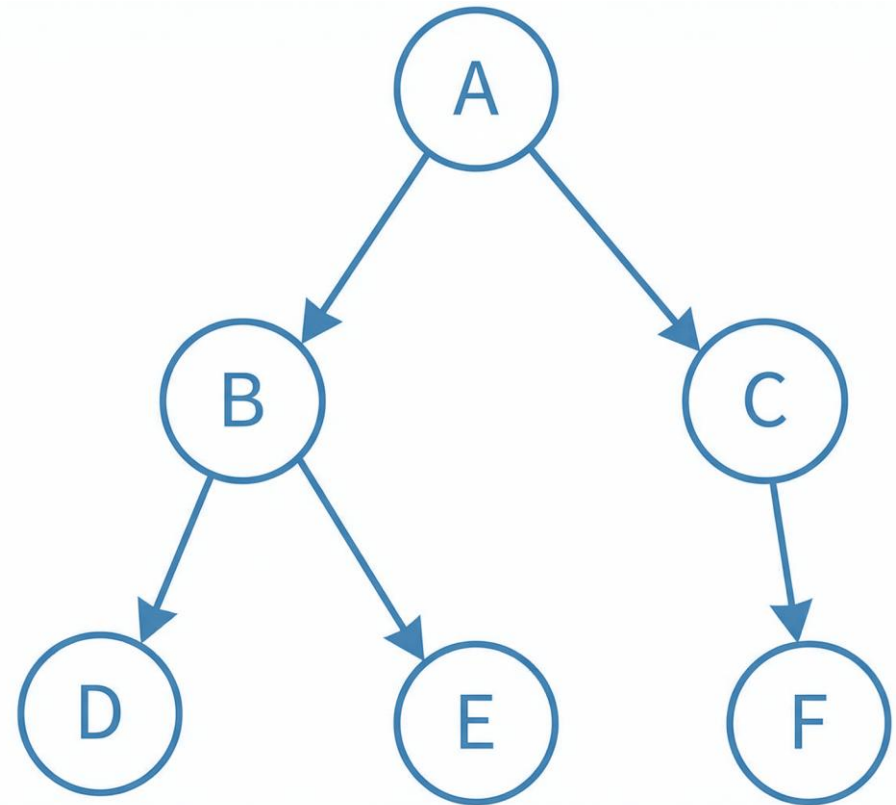
# Depth-First Search (DFS)

---

## ❖ Algorithm Logic:

Similar to BFS, but uses a **Stack** instead of a Queue.

1. Initialize all nodes to **Ready (1)**
2. Push start node A to Stack, set to **Waiting (2)**.
3. Repeat until Stack is empty:
  - a) Pop top node N. Process it. Set to **Processed (3)**.
  - b) Push neighbors of N that are **Ready (1)** to Stack, set to **Waiting (2)**.



## Depth-First Search



Here's the simulated Depth-First Search (DFS) on the provided graph:

## DFS Simulation Steps:

- **Initialization:** Stack: [], Visited: {}.

**1. Start at A:** Push A.

❖ Stack: [A], Visited: {A}.

**2. Process A, Move to B:** Pop A. Push A's unvisited neighbors (e.g., C, then B). Pop B. Push B's unvisited neighbors (e.g., E, then D).

❖ Stack: [C, D, E], Visited: {A, B, C, D, E}.

**3. Process D:** Pop D. D has no unvisited neighbors.

❖ Stack: [C, E].

**4. Process E:** Pop E. E has no unvisited neighbors

❖ Stack: [C].

**5. Process C, Move to F:** Pop C. Push C's unvisited neighbor (F).

❖ Stack: [F], Visited: {A, B, C, D, E, F}.

**6. Process F:** Pop F. F has no unvisited neighbors.

❖ Stack: [] (Empty).

**Final DFS Traversal Order:  $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F$**

# Differences Between BFS and DFS

Feature	BFS (Breadth First Search)	DFS (Depth First Search)
Traversal Method	Level-by-level (horizontal movement)	Depth-wise (vertical movement)
Data Structure	Queue	Stack / Recursion
Starting Node	Start from a source and explores neighbors first	Starts from a source and goes deep before backtracking
Uses	Shortest Path Finding, Cycle Detection, Connected Components, Network Routing	Detect cycles, path checking, topological sorting
Memory Usage	High (stores many nodes in queue)	Low (stores only path nodes)
Time Complexity	$O(V + E)$	$O(V + E)$
Space Complexity	$O(V)$	$O(V)$
Tree Produced	BFS Tree	DFS Tree
Nature of Search	Complete Search	Backtracking-based search

The background features a complex arrangement of overlapping triangles in various shades of blue, ranging from light sky blue to deep navy blue. These triangles are positioned primarily on the right side and bottom of the frame, creating a modern, geometric aesthetic. The rest of the background is a solid, very light blue.

# Thank You

For your patience