📘 GRAPH ALGORITHMS – DETAILED REVISION NOTES (GATE 2026)

1️⃣ Graph Traversal – Basic Concept

Graph Traversal is the process of visiting **each vertex exactly once** in some specific order and processing its information.

👉 Traversal is needed because:

- Graph has **no fixed root**
- Can be **connected or disconnected**

Types of Traversal

1. **Breadth First Search (BFS / BFT)**
2. **Depth First Search (DFS / DFT)**

2️⃣ Breadth First Search (BFS / BFT)

🔹 Core Idea

- Nodes are visited **level by level**
- All neighbours first, then neighbours of neighbours

🔹 Data Structure Used

- **Queue (FIFO)**

🔹 BFS Algorithm (Exam Writing)

1. Start from source vertex s
2. Mark s as visited and push into queue
3. While queue is not empty:
    - Dequeue a vertex u
    - Visit all **unvisited adjacent vertices** of u
    - Mark them visited and enqueue

🔹 Types of BFS

1. **FIFO BFS (Standard – GATE relevant)**
2. **LIFO BFS (Bonus concept)**
    - Uses queue but follows LIFO order
    - ❌ NOT same as DFS

🔹 Important Property (VERY IMPORTANT)

👉 In an **unweighted graph (or equal weight edges)**:

> **BFS finds Single Source Shortest Path (SSSP)**
> from source to all other vertices.

✔ Because BFS explores nodes in increasing order of distance.

📌 **GATE Trick:**
Shortest path + no weights given → **BFS**

🔹 BFS Complexity

Depends on representation:

Representation	Time Complexity
Adjacency Matrix	$O(V^2)$
Adjacency List	$O(V + E)$

🔹 Applications of BFS

- Shortest path in unweighted graph
- Connected components
- Bipartite graph checking
- Level order traversal

3️⃣ Depth First Search (DFS / DFT)

🔹 Core Idea

- Explore **as deep as possible**
- Backtrack when no unvisited neighbour

🔹 Data Structure Used

- **Stack / Recursion**

🔹 DFS Algorithm (Exam Writing)

1. Start from a vertex v
2. Mark v as visited
3. For each unvisited neighbour u of v:
    - Call DFS(u)
4. Continue until all reachable vertices are explored

4️⃣ DFS on Undirected Graph

Case 1: Connected Graph

- Single DFS call visits all vertices
- Produces **DFS Spanning Tree**

Case 2: Disconnected Graph

- Multiple DFS calls needed
- Produces **DFS Forest**
- Number of DFS calls = number of connected components

5️⃣ DFS Node Terminology (Exam Favorite)

During DFS, each node can be in one of three states:

1. **Exploring Node**
    - Node currently being processed
    - Present at top of stack
2. **Live Node**
    - Discovered but not fully explored
    - Present somewhere in stack
3. **Dead Node**
    - Fully explored
    - Removed (popped) from stack

6️⃣ DFS Timing Values (VERY VERY IMPORTANT)

For each vertex v:

🔹 Discovery Time `d(v)`

- Time when vertex is **visited first time**

🔹 Finishing Time `f(v)`

- Time when vertex becomes **dead node**
- All its adjacent vertices are explored

📌 These times are used in:

- Edge classification
- Topological sorting
- SCC properties
- PYQ logical questions

7️⃣ DFS on Directed Graph – Types of Edges

When DFS is applied on **directed graph**, edges are classified as:

1. Tree Edge

- Part of DFS spanning tree

2. Forward Edge

- From a node to its **non-child descendant**

3. Backward Edge

- From a node to its **ancestor**
- Indicates **cycle**

4. Cross Edge

- Between two nodes with **no ancestor–descendant relation**

🔹 Interval Property (GATE GOLD)

For any two vertices `u` and `v`, exactly one is true:

1. `[d(u), f(u)]` and `[d(v), f(v)]` disjoint
   → Cross edge
2. One interval inside another
   → Ancestor–descendant relation
   - Forward / Backward edge

8️⃣ Cycle Detection using DFS

- **Undirected Graph**
  - Back edge to visited node (not parent) → cycle
- **Directed Graph**
  - Presence of **backward edge** → cycle

📌 **Important Result**

> DAG never has backward edge

9️⃣ Directed Acyclic Graph (DAG)

Definition

A **directed graph with no cycles**

Properties

- No backward edges
- Topological sorting possible
- Used in scheduling, dependency resolution

🔟 Topological Sorting

🔹 Definition

Linear ordering of vertices such that:

> If there is an edge `u → v`, then `u` appears before `v`

🔹 Conditions

- Possible **ONLY for DAG**
- If cycle exists → topological sort ❌

🔹 Topological Sort using DFS (GATE Standard)

Algorithm:

1. Perform DFS on graph
2. Store vertices in order of finishing times
3. Arrange vertices in **descending order of finishing time**

📌 Vertex with **highest finishing time comes first**

1️⃣1️⃣ Connected Components (CC)

Definition

A **connected component** is a maximal connected subgraph.

Key Points

- Applies to **undirected graphs**
- Single isolated vertex = one CC
- Found using DFS / BFS

1️⃣2️⃣ Strongly Connected Components (SCC)

Definition

In a directed graph, vertices u and v are strongly connected if:

- Path from u → v
- Path from v → u

Properties (EXTREMELY IMPORTANT)

1. SCCs are **disjoint**
2. Every directed graph becomes a **DAG of SCCs**
3. If edge exists from SCC C → C′
   then **max finishing time of C > C′**

📌 This property is directly asked in GATE.

1️⃣3️⃣ Articulation Point (Cut Vertex)

Definition

A vertex whose removal:

- Increases number of connected components

Key Points

- Applies to **undirected graph**
- If graph has AP → graph is **not biconnected**

1️⃣4️⃣ Biconnected Graph & Biconnected Components (BCC)

Biconnected Graph

- Graph with **no articulation point**

Biconnected Component

- Maximal biconnected subgraph

Properties

- If AP count = 0 → graph itself is one BCC
- If AP exists → multiple BCCs

🔥 FINAL ONE-LOOK REVISION TABLE

- BFS → Queue → Shortest path (unweighted)
- DFS → Stack → Structure, cycle, SCC
- Backward edge → Cycle
- DAG → No backward edge
- Topological sort → Descending finishing time
- CC → Undirected
- SCC → Directed + mutual reachability
- AP → Removal disconnects graph
- BCC → No AP
