1. What is a Graph Database?

- A database that uses a **graph data structure** to model data.
- Composed of **nodes** and **edges**:
    - **Nodes** represent entities.
    - **Edges** represent relationships between entities.
- Each node and edge is uniquely identified and can have **properties** (e.g., name, type).
- Graph databases support graph-specific queries like:
    - **Traversals**
    - **Shortest path**
    - Many other advanced operations

2. Where Do Graphs Show Up?

- **Social Networks** – Instagram, Facebook, etc.
- **Sociology/Psychology** – Mapping human interactions
- **The Web** – Pages (nodes) and hyperlinks (edges)
- **Chemistry & Biology** – Molecular interactions, genetics, systems biology

3. Graph Basics (Labeled Property Graph)

- A graph is made of **nodes (vertices)** and **relationships (edges)**.
- **Labels** group nodes into categories (e.g., `person`, `car`).
- **Properties** are key-value pairs on nodes and edges.
- A node can exist without edges.
- An edge must always connect two nodes.

4. Example Structure

- **Labels**: `person`, `car`
- **Relationship types**: `Drives`, `Owns`, `Lives_with`, `Married_to`
- **Properties**: Custom key-value data on nodes/edges

5. What is a Path?

- A **path** is an ordered sequence of nodes connected by edges with **no repeats**.
- Example of a valid path: 1 → 2 → 6 → 5
- Invalid path (repeats a node): 1 → 2 → 6 → 2 → 3

6. Flavors of Graphs

- **Connected**: A path exists between every pair of nodes
- **Disconnected**: Not all nodes are reachable from each other
- **Weighted**: Edges have weights (e.g., cost, distance)
- **Unweighted**: Edges are equal
- **Directed**: Edges have a direction (start → end)
- **Undirected**: Edges have no direction
- **Acyclic**: No cycles
- **Cyclic**: Contains one or more cycles
- **Sparse**: Few edges relative to number of nodes
- **Dense**: Many edges relative to number of nodes

7. Trees

- A special kind of **acyclic** and **connected** graph.
- Often used in hierarchical modeling (e.g., file systems, org charts)

8. Graph Algorithms – Pathfinding

- **Pathfinding** is the most common graph operation.
- Goal: Find the **shortest path** between two nodes (fewest edges or lowest weight).
- Use cases:
    - Efficiency/resiliency analysis (e.g., **Average Shortest Path**)
    - **Minimum Spanning Tree**, **Cycle Detection**, **Max/Min Flow**

9. BFS vs DFS

- **BFS (Breadth-First Search)**: Explores neighbors level by level
- **DFS (Depth-First Search)**: Explores as far as possible along each branch

10. Shortest Path Algorithms

- Used to find minimum distance or cost between two nodes.
- May involve edge weights.

11. Centrality & Community Detection

Centrality

- Identifies the **most influential/important** nodes in a graph
- Example: Finding social media influencers

Community Detection

- Finds **clusters** or **partitions** in a graph
- Reveals sub-groups and structural relationships

12. Famous Graph Algorithms

- **Dijkstra's Algorithm**: Finds shortest paths in a graph with positive weights
- **A***: Enhances Dijkstra's with heuristics to guide the search
- **PageRank**: Ranks importance of nodes based on incoming links and their importance

13. Neo4j – A Graph Database System

- Supports **transactional** and **analytical** graph queries
- Schema-optional design
- ACID-compliant
- Distributed computing support
- Indexing supported
- Other similar systems: Microsoft CosmosDB, Amazon Neptune