UNIT IV: Graphs

A Comprehensive Study Guide with Real-Time Examples

Graph Concepts

A graph is a non-linear data structure consisting of vertices (nodes) and edges (connections). Graphs model many real-world scenarios with relationships between entities.

Basic Terminology:

Vertex/Node: Individual element in the graph

Edge: Connection between two vertices

Degree: Number of edges connected to a vertex

Path: Sequence of vertices connected by edges

Cycle: Path that starts and ends at the same vertex

Types of Graphs:

Directed Graph (Digraph): Edges have direction (A→B means one-way)

Undirected Graph: Edges have no direction (A-B means bidirectional)

Weighted Graph: Edges have weights (distances, costs)

Unweighted Graph: All edges have equal weight (typically 1)

Real-Time Example:

Social Network: Facebook connections form an undirected graph. Each user is a vertex, and friendships are undirected edges. Instagram followers form a directed graph where A→B means A follows B (but B may not follow A). When Facebook recommends friends, it traverses the graph.

Graph Representation

Graphs can be represented in memory using different approaches, each with trade-offs.

1. Adjacency Matrix:

A 2D array where matrix[i][j] = 1 if there's an edge from vertex i to j, 0 otherwise.

For weighted graphs, matrix[i][j] = weight of edge, or ∞ if no edge exists.

Space Complexity: $O(V^2)$

Edge Lookup: O(1)

Best for: Dense graphs, frequent edge queries

2. Adjacency List:

An array of linked lists where each vertex maintains a list of its adjacent vertices.

Space Complexity: O(V + E)

Edge Lookup: O(degree of vertex)

Best for: Sparse graphs, memory-efficient storage

Real-Time Example:

GPS Navigation: Maps are represented as graphs. Adjacency lists store roads connected to each intersection. When calculating routes, the algorithm traverses from your location through nearby intersections, making adjacency lists more practical than matrices (since not every intersection connects to every other).

Graph Traversals

Methods to systematically visit all vertices in a graph.

1. Breadth-First Search (BFS):

Explores vertices level by level, starting from a source vertex. Uses a queue.

Time Complexity: O(V + E)

Space Complexity: O(V)

Properties: Finds shortest path in unweighted graphs, explores nearest vertices first

Real-Time Example:

Social Media Friend Recommendations: When you search for friends on LinkedIn, BFS explores your connections (distance 1), then their connections (distance 2), and recommends people at distance 2 or 3. This ensures recommendations are from relatively close social networks.

2. Depth-First Search (DFS):

Explores vertices by going as deep as possible before backtracking. Uses a stack or recursion.

Time Complexity: O(V + E)

Space Complexity: O(V)

Properties: Useful for detecting cycles, finding connected components, topological sorting

Real-Time Example:

Puzzle Solving: Games like Sudoku or maze solving use DFS. The algorithm chooses one path, goes as deep as possible, and backtracks when hitting a dead end. This explores all possible solutions until finding a valid one.

Shortest Path Algorithms

1. Dijkstra's Algorithm:

Finds the shortest path from a source vertex to all other vertices in weighted graphs with non-negative edge weights.

Time Complexity: $O((V + E) \log V)$ with min heap

Greedy Approach: Always selects the unvisited vertex with smallest distance

Real-Time Example:

GPS Navigation: When calculating the shortest route from point A to point B, GPS uses Dijkstra's algorithm. The graph represents intersections (vertices) and roads (weighted edges with distances). The algorithm finds the path minimizing travel distance or time.

2. Bellman-Ford Algorithm:

Finds shortest paths allowing negative edge weights but no negative cycles.

Time Complexity: $O(V \times E)$

Advantage: Handles negative weights

Real-Time Example:

Currency Exchange: When converting currencies, negative weights represent gaining money in certain exchange paths. Bellman-Ford detects arbitrage opportunities (negative cycles) in forex markets.

3. Floyd-Warshall Algorithm:

Finds shortest paths between all pairs of vertices.

Time Complexity: $O(V^3)$

Use: Small graphs where all-pairs distances needed

Minimum Spanning Tree

A spanning tree of a graph is a subgraph that includes all vertices and is a tree (acyclic). A minimum spanning tree (MST) has the minimum total edge weight.

1. Kruskal's Algorithm:

Builds MST by sorting edges by weight and greedily adding edges that don't form cycles.

Time Complexity: O(E log E)

Data Structure: Uses Union-Find to detect cycles

2. Prim's Algorithm:

Builds MST by starting from a vertex and repeatedly adding the smallest edge connecting the tree to a new vertex.

Time Complexity: O((V + E) log V) with min heap

Approach: Greedy, vertex-centric

Real-Time Example:

Network Design: Suppose you're connecting 5 cities with telecommunication cables. Each potential connection has a cost (weight). MST algorithms find the minimum cost network connecting all cities. Kruskal or Prim's would be used to connect cities with minimal total cable length, saving on infrastructure costs.

Another Real-Time Example:

Airport Hub Design: Airlines use MST to design hub networks. Vertices represent cities, edges are flight routes with costs. MST ensures all cities are connected with minimum operational cost while maintaining connectivity.