***Patient Management System Using Graphs***

**1. Introduction**

Healthcare systems require efficient data management for handling patient records, referrals, and relationships. This program implements a graph-based approach to manage patient interactions, making it easier to store, retrieve, and analyze medical connections dynamically.

Instead of using traditional databases or lists, this system treats patients as nodes and their relationships as edges, allowing for flexible and efficient querying.

**2. System Design & Data Structures**

**A**. Graph Representation (Adjacency List Approach)

Each patient is stored as a node in a graph.

Connections (edges) between patients indicate

relationships such as family ties, referrals, or shared medical conditions.

The adjacency list representation is implemented using an unordered\_map:

Key → Patient ID

Value → List of {Neighbor ID, Relationship Weight} pairs

Example Representation

If the system has the following relationships:

Patient 101 is connected to 102 (doctor referral).

Patient 102 is connected to 103 (family member).

Patient 101 is also connected to 104 (friend).

The adjacency list would look like this:

101 → [(102, 1), (104, 2)]

102 → [(103, 1)]

103 → []

104 → []

Here, the weight represents different types of relationships (e.g., 1 = doctor referral, 2 = friend).

**B**. Data Structures Used

The adjacency list is used instead of an adjacency matrix because:

It saves memory (only stores actual connections).

It allows fast lookups (O(1) for checking a patient’s existence).

It scales better for large numbers of patients.

**3. Functionalities & Algorithms**

A. Adding and Deleting Patients

1. Add a Patient

If the patient does not exist, a new entry is created in the adjacency list.

If the patient already exists, an error message is shown.

2. Delete a Patient

The patient is removed from the adjacency list.

All connections to this patient are removed from other patients.

Efficiency

Adding a patient → O(1)

Deleting a patient → O(n), since all references to the patient must be removed.

B. Managing Connections

3. Add a Connection Between Patients

Establishes a directed edge (one-way connection) or undirected edge (mutual relationship).

A weight may be added, e.g., 1 = family, 2 = doctor referral, etc..

**4. Remove a Connection**

Deletes the edge between two patients.

Efficiency

Adding/removing connections → O(1)

C. Searching for Patients

5. Find a Patient

Checks if a patient exists in the system.

Returns a list of all directly connected patients.

6. Finding All Related Patients (Graph Traversal Algorithms)

Breadth-First Search (BFS):

Used when finding all patients directly or indirectly connected to a given patient.

Use Case: Finding all potential contacts of an infected patient.

Depth-First Search (DFS):

Used for deep exploration of relationships (e.g., tracing medical histories).

**Efficiency**

BFS & DFS → O(V + E), where V = number of patients and E = number of connections.

4. Additional Features & Enhancements

A. Sorting & Prioritization

Patients can be prioritized based on medical conditions.

Sorting connections based on urgency levels (e.g., critical patients first).

B. Pathfinding Between Patients

Dijkstra’s Algorithm can be implemented to find

the shortest referral chain between two patients.

Use Case: Finding the shortest doctor referral path from one specialist to another.

C. Predictive Analytics (AI Integration)

The system can be enhanced to analyze patient patterns.

Machine Learning can be applied to predict potential disease transmission among connected patients.

**5. Real-World Applications**

**6. Efficiency Considerations & Scalability**

A. Time Complexity Analysis

B. Scalability Strategies

1. Use a Database Backend (SQL/NoSQL)

Instead of using an in-memory adjacency list, store patients in a graph database (e.g., Neo4j for highly connected data).

2. Distributed Processing for Large Data

If the number of patients is very large (millions), use parallel computing to distribute graph queries across multiple servers.

3. Graph Partitioning

Divide the graph into subgraphs based on hospital regions to improve performance.

**Conclusion**

This program efficiently manages patient relationships using graph-based structures, providing a scalable and efficient solution for healthcare management. By leveraging graph algorithms, real-world applications, and future enhancements, this system can be expanded

into a fully functional patient-tracking solution.