### **Hospital Management System Documentation ( Code )**

#### **Objective:**

The **Hospital Management System (HMS)** is designed to efficiently manage patient records, appointments, and emergency cases. The system uses multiple data structures including **Binary Search Tree (BST)**, **AVL Tree**, **Queue**, and **Stack** to store and manage patient data effectively.

### **Key Features:**

1. **Patient Registration (BST):**
   * The HMS uses a **Binary Search Tree (BST)** to store patient records.
   * Each node in the BST represents a patient and stores the patient's ID, name, age, and medical history.
   * The BST ensures efficient operations like insertion, deletion, and retrieval of patient records based on patient ID.
2. **Balanced Data Storage (AVL Tree):**
   * An **AVL Tree**, a self-balancing binary search tree, is used to maintain a balanced dataset for patient records.
   * This structure ensures that operations remain efficient even as the number of patients increases, maintaining O(log n) complexity for insertion, deletion, and lookup operations.
   * The AVL Tree is crucial for storing critical patient data that requires frequent access and updates.
3. **Appointment Scheduling (Queue):**
   * A **Queue** data structure is employed to manage patient appointments.
   * Patients are added to the queue based on their appointment schedule.
   * The queue provides methods to add a new appointment, check the next patient in line, and remove patients who have completed their appointments.
4. **Emergency Handling (Stack):**
   * Emergency cases are managed using a **Stack**.
   * The stack follows a **Last In, First Out (LIFO)** principle, ensuring that the most recent emergency is addressed first.
   * Patients in emergency situations are pushed onto the stack, and the system can quickly access and handle these cases as they arise.
5. **Patient Check-in and Check-out (Combination of Data Structures):**
   * Upon check-in, patient details are added to both the BST and AVL Tree.
   * During check-out, the patient's record is either updated or removed from both trees.
   * The system also manages the queue and stack for appointments and emergency cases respectively.

### **Implementation Details:**

1. **Data Structures:**
   * **BST** and **AVL Tree** classes are implemented with essential methods such as insert, delete, search, rotate, and more.
   * **Queue** and **Stack** classes are implemented with standard operations like enqueue, dequeue, push, pop, and so forth.
2. **Classes and Methods:**
   * **Patient:** A class that stores patient details including ID, name, age, and medical history.
   * **HospitalManagementSystem:** This class integrates all operations and manages patient registration, check-in, check-out, appointment scheduling, and emergency handling.
     + registerPatient(patient): Registers a new patient in the system.
     + checkInPatient(patient): Checks in a patient by adding their details to the BST and AVL Tree.
     + checkOutPatient(patient\_id): Checks out a patient, removing their details from the BST and AVL Tree.
     + scheduleAppointment(patient): Adds a patient to the appointment queue.
     + handleEmergency(patient): Adds a patient to the emergency stack.
     + getNextAppointment(): Retrieves the next patient from the appointment queue.
     + getNextEmergency(): Retrieves the next patient from the emergency stack.
     + searchPatient(patient\_id): Searches for a patient in the BST and AVL Tree using their ID.
3. **User Interface:**
   * A console-based UI is provided for interaction, offering options to register patients, schedule appointments, handle emergencies, check in/out patients, and search for patient records.

Example 👍:



### **Additional Features:**

* **Priority Queue:** Implement a priority queue to handle patients based on the severity of their conditions.
* **Database Integration:** Persist patient records beyond the application's runtime by integrating with a database.
* **Web Interface:** A web-based interface to enhance user interaction.

### **Benefits:**

* Efficient management of patient records with balanced data structures.
* Quick retrieval and updating of patient information.
* Effective handling of emergency cases and appointments.
* Demonstrates practical use of multiple data structures in a real-world scenario.

**Conclusion 👍:**

This project showcases a comprehensive approach to managing hospital operations, leveraging data structures for efficient and effective handling of patient records and processes.