Project: Hospital Management System

Objective:

Develop a Hospital Management System to efficiently manage patients' records, including check-ins, check-outs, and retrieval of patient details, using BST, AVL Tree, Stack, and Queue data structures.

Key Features:

1. **Patient Registration (BST):**
    - Use a Binary Search Tree to store patient records.
    - Each node will contain patient information such as ID, name, age, and medical history.
    - The BST allows for efficient insertion, deletion, and retrieval of patient records based on patient ID.
2. **Balanced Data Storage (AVL Tree):**
    - Implement an AVL Tree to keep the patient records balanced.
    - This ensures that operations like insertion, deletion, and lookup remain efficient even as the number of patients grows.
    - Use the AVL Tree for storing and balancing critical patient data that requires frequent access and updates.
3. **Appointment Scheduling (Queue):**
    - Use a Queue to manage patient appointments.
    - Patients are added to the queue based on their appointment times.
    - Implement functionality to add a new appointment, check the next patient in line, and remove patients who have completed their appointments.
4. **Emergency Handling (Stack):**
    - Use a Stack to manage emergency cases.
    - When an emergency patient arrives, their details are pushed onto the stack.
    - Emergency patients are handled in a LIFO (Last In, First Out) order, ensuring that the most recent emergency is addressed first.
5. **Patient Check-in and Check-out (Combination of Data Structures):**
    - Upon check-in, patient details are added to the BST and AVL Tree.
    - During check-out, the patient's record is updated or removed from both trees.
    - Manage the queue and stack for appointment and emergency cases accordingly.

Implementation Details:

1. **Data Structures:**
    - Implement BST and AVL Tree classes with necessary methods (insert, delete, search, rotate, etc.).
    - Implement Queue and Stack classes with standard operations (enqueue, dequeue, push, pop, etc.).
2. **Classes and Methods:**

    - `Patient`: A class to store patient details.
    - `HospitalManagementSystem`: A class to manage all operations, including:
        - `registerPatient()`: Register a new patient.
        - `checkInPatient()`: Check-in a patient.
        - `checkOutPatient()`: Check-out a patient.
        - `scheduleAppointment()`: Add a patient to the appointment queue.
        - `handleEmergency()`: Add a patient to the emergency stack.
        - `getNextAppointment()`: Get the next patient from the appointment queue.
        - `getNextEmergency()`: Get the next emergency patient from the stack.
        - `searchPatient()`: Search for a patient using the BST and AVL Tree.
3. **User Interface:**
    - Create a simple console-based UI for interacting with the system.
    - Provide options for registering patients, scheduling appointments, handling emergencies, checking in/out patients, and searching for patient records.

Additional Features (Optional):

- **Priority Queue:** Implement a priority queue for handling patients based on the severity of their conditions.
- **Database Integration:** Integrate with a database to persist patient records beyond the application's runtime.
- **Web Interface:** Develop a web-based interface for better 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.
- Practical use of multiple data structures in a real-world scenario.

This project not only demonstrates your understanding of various data structures but also provides a useful application that can be expanded and improved over time.