Hospital Management System Using AVL Tree

1. Overview

The Hospital Management System (HMS) is a software solution designed to manage patient records, appointments, and emergency handling efficiently. This system utilizes an **AVL Tree** for maintaining patient records, ensuring that the data remains balanced and retrieval operations are performed quickly. Additionally, **Queue and Stack** data structures are used for managing patient appointments and emergency cases, respectively.

2. Key Components

1. AVL Tree :

A self-balancing binary search tree that ensures **O(log n)** time complexity for insertion, deletion, and search operations. This is crucial for efficient management and retrieval of patient records.

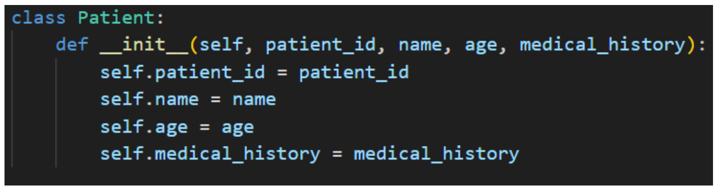
2. Queue : A FIFO (First-In-First-Out) data structure used to manage patient appointments in the order they are scheduled.

3. Stack : A LIFO (Last-In-First-Out) data structure used to handle emergency cases, where the most recent case is handled first.

3. Class Implementations

3.1 Patient Class

The Patient class represents an individual patient in the system with basic details like patient_id, name, age, and medical_history.



```python
class Patient:
    def __init__(self, patient_id, name, age, medical_history):
        self.patient_id = patient_id
        self.name = name
        self.age = age
        self.medical_history = medical_history
```

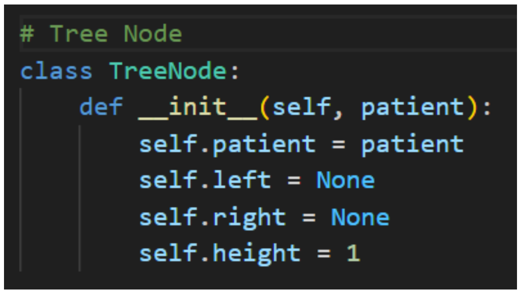
Attributes:

- **patient_id** : Unique identifier for the patient.
- **name** : Name of the patient.
- **age** : Age of the patient.

- **medical_history**: Medical history or ailment details of the patient.

3.2 TreeNode Class

The TreeNode class is a helper class for the AVL Tree. It represents a node in the AVL tree containing a patient and pointers to its left and right children.



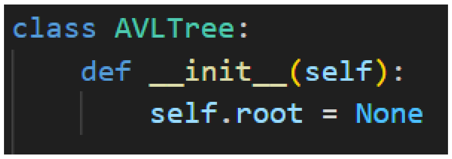
```python
# Tree Node
class TreeNode:
    def __init__(self, patient):
        self.patient = patient
        self.left = None
        self.right = None
        self.height = 1
```

Attributes:

- patient: A Patient object stored in the node.
- left: Reference to the left child node.
- right: Reference to the right child node.
- height: Height of the node in the AVL tree (used for balancing).
-

3.3 AVLTree Class

The AVLTree class implements all the operations required to manage a balanced AVL Tree. It inherits basic tree functionalities and adds balancing logic.



```python
class AVLTree:
    def __init__(self):
        self.root = None
```

Attributes:

- **root**: Root node of the AVL tree.

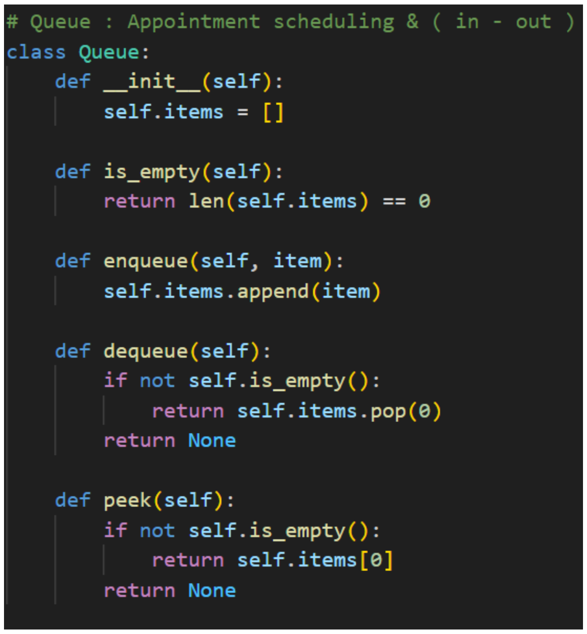
Methods:

- **insert(root, patient)**: Inserts a new Patient into the AVL tree while maintaining balance.

- **delete(root, patient_id)**: Deletes a Patient from the AVL tree by patient_id while maintaining balance.
- **left_rotate(z)**: Performs a left rotation around node z.
- **right_rotate(z)**: Performs a right rotation around node z.
- **get_height(root)**: Returns the height of a node.
- **get_balance(root)**: Computes the balance factor of a node.
- **get_min_value_node(root)**: Finds the node with the minimum patient_id in the subtree rooted at root.
- **search(root, patient_id)**: Searches for a patient in the AVL tree by patient_id.

3.4 Queue Class

The Queue class is used to manage appointments. It implements basic queue operations.



```
# Queue : Appointment scheduling & ( in - out )
class Queue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def peek(self):
        if not self.is_empty():
            return self.items[0]
        return None
```

Attributes:

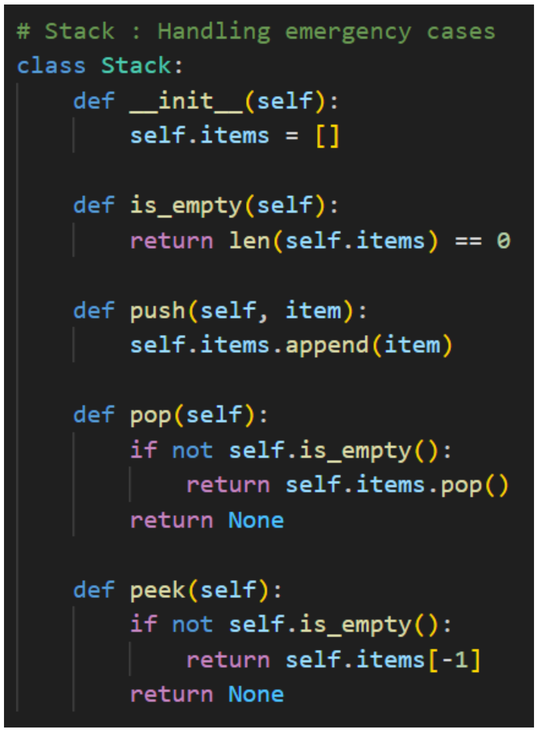
- **items:** A list to store queue elements.

Methods:

- **is_empty()**: Checks if the queue is empty.
- **enqueue(item)**: Adds an item to the end of the queue.
- **dequeue():** Removes and returns the item from the front of the queue.
- **peek()**: Returns the item at the front of the queue without removing it.
-

3.5 Stack Class

The Stack class is used to manage emergency cases. It implements basic stack operations.



```python
# Stack : Handling emergency cases
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None

    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        return None
```

Attributes:

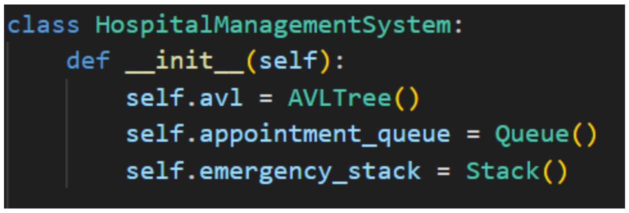
- **items**: A list to store stack elements.

Methods:

- **is_empty()**: Checks if the stack is empty.
- **push(item)**: Adds an item to the top of the stack.
- **pop()**: Removes and returns the item from the top of the stack.
- **peek():** Returns the item at the top of the stack without removing it.

3.6 Hospital Management System Class

The Hospital Management System class integrates all components and provides an interface to manage patient records, appointments, and emergencies.



```python
class HospitalManagementSystem:
    def __init__(self):
        self.avl = AVLTree()
        self.appointment_queue = Queue()
        self.emergency_stack = Stack()
```

Attributes:

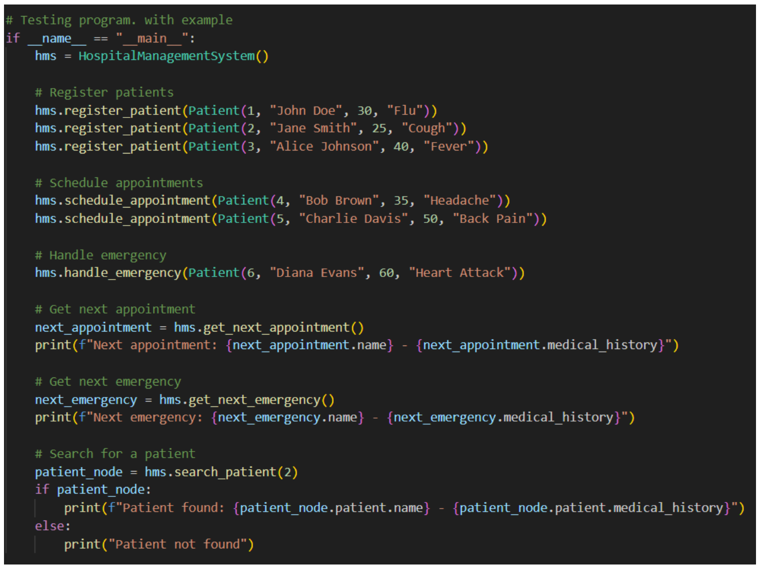
- **avl**: An instance of AVLTree to manage patient records.
- **appointment_queue**: An instance of Queue to manage patient appointments.
- **emergency_stack**: An instance of Stack to manage emergency cases.

Methods:

- **register_patient(patient)**: Registers a new patient in the AVL tree.
- **check_in_patient(patient)**: Checks in a patient by registering them.
- **check_out_patient(patient_id)**: Checks out a patient by deleting them from the AVL tree.
- **schedule_appointment(patient)**: Schedules a patient appointment by adding them to the queue.
- **handle_emergency(patient)**: Handles an emergency case by pushing the patient onto the stack.
- **get_next_appointment()**: Retrieves the next appointment from the queue.
- **get_next_emergency()**: Retrieves the next emergency case from the stack.
- **search_patient(patient_id)**: Searches for a patient in the AVL tree by patient_id.

4. Example

The following example demonstrates how to use the HospitalManagementSystem class to manage patients, appointments, and emergencies:



```python
# Testing program. with example
if __name__ == "__main__":
    hms = HospitalManagementSystem()

    # Register patients
    hms.register_patient(Patient(1, "John Doe", 30, "Flu"))
    hms.register_patient(Patient(2, "Jane Smith", 25, "Cough"))
    hms.register_patient(Patient(3, "Alice Johnson", 40, "Fever"))

    # Schedule appointments
    hms.schedule_appointment(Patient(4, "Bob Brown", 35, "Headache"))
    hms.schedule_appointment(Patient(5, "Charlie Davis", 50, "Back Pain"))

    # Handle emergency
    hms.handle_emergency(Patient(6, "Diana Evans", 60, "Heart Attack"))

    # Get next appointment
    next_appointment = hms.get_next_appointment()
    print(f"Next appointment: {next_appointment.name} - {next_appointment.medical_history}")

    # Get next emergency
    next_emergency = hms.get_next_emergency()
    print(f"Next emergency: {next_emergency.name} - {next_emergency.medical_history}")

    # Search for a patient
    patient_node = hms.search_patient(2)
    if patient_node:
        print(f"Patient found: {patient_node.patient.name} - {patient_node.patient.medical_history}")
    else:
        print("Patient not found")
```

The Hospital Management System implemented with an AVL Tree, Queue, and Stack provides a robust solution for managing patient records efficiently. The AVL tree ensures that patient records are maintained in a balanced manner, allowing quick insertions, deletions, and searches. The Queue and Stack effectively manage appointments and emergency cases, respectively, enabling a streamlined workflow in a hospital setting.

This code is designed to be modular and extendable, allowing for additional features to be added with ease, such as advanced search filters, patient history management, and integration with other hospital systems.