DS2030 Data Structures and Algorithms for Data Science
Lab 3 (Take Home) Due on August 28, 11.59pm

Instructions

- You are to use Python as the programming language. You may use Visual Studio Code (or any other editor you are comfortable with) as the IDE.
- You have to work individually for this lab.
- You are not allowed to share code with your classmates nor allowed to use code from the internet. You are encouraged to engage in high-level discussions with your classmates; however, ensure to include their names in the report/code documentation. If you refer to any source on the Internet, include the corresponding citation in the report/code documentation. If we find that you have copied code from your classmate or from the Internet, you will receive a failing grade in the course.
- The submission must be a zip file with the following naming convention: `rollnumber.zip`. The Python files should be contained in a folder named after the question number.
- Include appropriate comments to document the code. Include a `readme` file containing instructions for executing the code. The code should run on institute Linux machines.
- Upload your submission to Moodle by the due date and time. Do not email the submission to the instructor or the TA.

This lab will improve your understanding of linkedlists and priority queues.

1 Lab Overview

In this lab, you will implement a **Patient Management System** for a clinic. The clinic has a limited number of doctors, and patients arrive with varying levels of severity and different condition types. Your goal is to manage the patient queue efficiently using a priority queue implemented with a linked list, prioritize patients based on severity, arrival time, and condition type, and ensure doctors are utilized optimally.

2 Objectives

- Implement a **Linked List** based **Priority Queue**.
- Manage the queue of patients and prioritize their treatment based on severity.
- Utilize doctors efficiently in treating patients.
- Understand and apply data structures like linked lists in real-world scenarios.

3 Step-by-Step Implementation Instructions

3.1 Step 1: Define the `Patient` Class

Objective: Define a class to represent a patient with attributes like name, severity, arrival time, and condition type.

1. Define the `Patient` class with the following attributes:
    - `name` (`str`): The name of the patient.
    - `severity` (`int`): The severity of the patient's condition. A numeric value representing the severity of the condition. Higher values indicate more severe conditions.
    - `arrival_time` (`int`): The time when the patient arrives at the clinic.
    - `type` (`str`): The type of condition (`"emergency"`, `"regular"`, or `"follow-up"`).
2. Add a method to determine the treatment time based on the condition type. The treatment times for the 'emergency', 'regular', and 'follow-up' are 5, 3, and 2 respectively.
3. Implement the comparison method (`__lt__`) in the `Patient` class to compare patients based on severity, arrival time, and condition type. Compare by severity first, followed by arrival time, and finally by condition type. If two patients have the same severity, prioritize by arrival time. Among patients with the same severity and arrival time, prioritize `"emergency"` patients over `"regular"` and `"follow-up"` patients.

```python
class Patient:
    # constructor for the class
    def __init__(self, name: str, severity: int, arrival_time: int,
        condition_type: str):
        # To do

    # estimate the treatment time
    def get_treatment_time(self) -> int:
        # To do

    # Implement comparison method for priority queue
    def __lt__(self, other: 'Patient') -> bool:
        # To do
```

Listing 1: Sample Patient Class

3.2 Step 2: Implement the `Node` Class

Objective: Define a node that will be used in the linked list. Each node will store a `Patient` and a reference to the next node.

1. Define a `Node` class that initializes with a `Patient` object and has a `next` pointer.

```python
class Node:
    def __init__(self, patient: 'Patient'):
        # To do
```

Listing 2: Sample Node Class

3.3 Step 3: Implement the `PriorityQueue` Class

Objective: Create a priority queue using a linked list where patients are ordered by their priority.

1. Define a `PriorityQueue` class with a `head` attribute that points to the start of the linked list.
2. Implement an `is_empty()` method to check if the queue is empty.
3. Implement the `insert(patient: Patient)` method to insert a new patient into the linked list in the correct priority order.

4. Implement the `pop()` method to remove and return the patient with the highest priority.

```python
class PriorityQueue:
    def __init__(self):
        # To do

    def is_empty(self) -> bool:
        # To do

    def insert(self, patient: 'Patient') -> None:
        # To do

    def remove(self) -> 'Patient':
        # To do
```

Listing 3: Sample PriorityQueue Class

3.4 Step 4: Implement the `Clinic` Class

Objective: Manage the clinic, including adding patients to the queue, treating patients, and managing doctor availability.

1. Define a `Clinic` class that initializes with the number of doctors and uses the `PriorityQueue` to manage patients.
2. Implement the `add_patient(patient: Patient)` method to add a patient to the queue.
3. Implement the `treat_patient()` method to assign the highest-priority patient to an available doctor and return the doctor and patient.
4. Implement the `update_doctor_availability()` method to update doctor availability based on the current time.

```python
class Clinic:
    def __init__(self, num_doctors: int):
        self.num_doctors = num_doctors
        self.doctors = [None] * num_doctors  # None means available
        self.patient_queue = PriorityQueue()
        self.current_time = 0

    def add_patient(self, patient: Patient) -> None:
        # To do

    def treat_patient(self) -> Tuple[Optional[str], Optional[Patient]]:
        # To do

    def update_doctor_availability(self) -> None:
        # To do
```

Listing 4: Sample Clinic Class

3.5 Step 5: Run the Simulation

Objective: Test the system with a sample set of patients and a clinic with a fixed number of doctors.

1. Create an instance of the `Clinic` class and add patients to it.

```
clinic = Clinic(num_doctors=3)

patients = [
    Patient("Aarav", 5, time.time(), "regular"),
    Patient("Isha", 7, time.time() + 1, "emergency"),
    Patient("Ravi", 6, time.time() + 2, "follow-up"),
    Patient("Meera", 4, time.time() + 3, "regular"),
    Patient("Aditya", 8, time.time() + 4, "emergency"),
]

for patient in patients:
    clinic.add_patient(patient)
```

Listing 5: Sample Simulation Code

1. Simulate the treatment process by updating doctor availability and treating patients over a fixed time period.

```
time_unit = 1
while time_unit <= 10:
    print(f"\nTime unit {time_unit}:")
    clinic.update_doctor_availability()
    doctor, patient = clinic.treat_patient()
    if doctor and patient:
        print(f"{doctor} is treating {patient.name} for {patient.type}.")
    time_unit += 1
    time.sleep(1)
```

Listing 6: Sample Simulation Loop