Data Structures: Queues Week Plan

Authors: Refat Othman and Diaeddin Rimawi

Lecture 1: Introduction to Queues (2 hours)

1. What is a Queue?

- Abstract data type representing a collection of elements.
- Follows the **FIFO** principle (First In, First Out).
- Operations:
  - **enqueue**: Add an element to the rear
  - **dequeue**: Remove the element from the front
  - **peek/front**: View the front element without removing it
- Real-life analogy: A queue in a supermarket or printer jobs in a spooler

2. Core Queue Operations (with visual examples)

- **Enqueue Example**:
  - Queue (front to rear): `1 <- 2` → enqueue(3) → `1 <- 2 <- 3`
- **Dequeue Example**:
  - Queue: `1 <- 2 <- 3` → dequeue() → `2 <- 3`
- **Peek Example**:
  - Queue: `2 <- 3` → peek() → `2`

3. Queue Implementations

- **Using a Linked List**:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedQueue:
    def __init__(self):
        self.front = None
        self.rear = None

    def enqueue(self, data):
        new_node = Node(data)
        if self.rear is None:
            self.front = self.rear = new_node
```

```
        else:
            self.rear.next = new_node
            self.rear = new_node

    def dequeue(self):
        if self.front is None:
            return None
        value = self.front.data
        self.front = self.front.next
        if self.front is None:
            self.rear = None
        return value

    def peek(self):
        return None if self.front is None else self.front.data

    def is_empty(self):
        return self.front is None

    def clear(self):
        self.front = self.rear = None
```

- **Using an Array**:

```
class ArrayQueue:
    def __init__(self, capacity):
        self.queue = [None] * capacity
        self.front = 0
        self.rear = 0
        self.capacity = capacity
        self.size = 0

    def is_empty(self):
        return self.size == 0

    def is_full(self):
        return self.size == self.capacity

    def enqueue(self, data):
        if self.is_full():
            print("Queue is full. Cannot enqueue.")
            return
        self.queue[self.rear] = data
        self.rear = (self.rear + 1) % self.capacity
        self.size += 1

    def dequeue(self):
        if self.is_empty():
            return None
        value = self.queue[self.front]
        self.front = (self.front + 1) % self.capacity
        self.size -= 1
```

```
        return value

    def peek(self):
        if self.is_empty():
            return None
        return self.queue[self.front]
```

4. Applications of Queue

4.1 Print Queue Management

```
print_queue = LinkedQueue()

def add_print_job(job):
    print_queue.enqueue(job)

def process_print_job():
    job = print_queue.dequeue()
    if job:
        print(f"Printing: {job}")

# Example:
add_print_job("Document1")
add_print_job("Document2")
process_print_job()
```

4.2 CPU Task Scheduling

```
tasks = LinkedQueue()

def add_task(task):
    tasks.enqueue(task)

def run_task():
    task = tasks.dequeue()
    if task:
        print(f"Running: {task}")

# Example:
add_task("TaskA")
add_task("TaskB")
run_task()
```

4.3 Call Center Support Line

```
support_queue = LinkedQueue()
```

```
def new_call(caller):
    support_queue.enqueue(caller)

def answer_call():
    caller = support_queue.dequeue()
    if caller:
        print(f"Answering call from: {caller}")

# Example:
new_call("Alice")
new_call("Bob")
answer_call()
```

Lecture 2: Exercises on Queues (2 hours)

1. Exercise 1: Implement FIFO using LIFO

Implement the First-In-First-Out (FIFO) concept using only the Last-In-First-Out (LIFO) concept.

2. Exercise 2: Implement LIFO using FIFO

Implement the Last-In-First-Out (LIFO) concept using only the First-In-First-Out (FIFO) concept.

3. Exercise 3: Time Needed to Buy Tickets

There are n people in a line queuing to buy tickets, where the 0th person is at the front of the line and the (n - 1)th person is at the back of the line.

You are given a 0-indexed integer array `tickets` of length n where the number of tickets that the ith person would like to buy is `tickets[i]`.

Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to go back to the end of the line (which happens instantaneously) in order to buy more tickets. If a person does not have any tickets left to buy, the person will leave the line.

Return the time taken for the person initially at position `k` (0-indexed) to finish buying tickets.

Example 1:

- Input: `tickets = [2,3,2], k = 2`
- Output: `6`

Example 2:

- Input: `tickets = [5,1,1,1], k = 0`
- Output: `8`

Assignment for Next Week

Task: Simulate a basic customer service system using a queue.

Customers arrive and join a service queue. Each customer is served in the order they arrive. Once served, they leave the queue. Your task is to:

- Enqueue customer names as they arrive.
- Dequeue and display the name of each customer as they are served.
- When the queue is empty, display a message indicating that all customers have been served.

Example:

```
# Sample Output:
Arriving: Alice
Arriving: Bob
Arriving: Carol
Serving: Alice
Serving: Bob
Serving: Carol
All customers served.
```