Project Title: Implementation and Application of Queue Data Structure

Submitted by:

AARUSH C S
ATHUL K KOSHY
MOHAMMED RIZWAN PP
MOIDEEN NIHAL

---

1. Introduction

The Queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. It is used in various real-life and computer science applications, such as CPU scheduling, printer spooling, and customer service systems. In this project, we explore how queues operate, implement them in Python, and apply them to simulate real-world tasks.

---

2. Problem Statement

Managing tasks or data in the order they arrive is crucial for fair and efficient processing. Without a structured system like a queue, operations can become disorganized, causing delays and inconsistencies. This project aims to design a queue system that simulates real-world scenarios like service lines or job processing systems.

---

3. Objectives

- **Understand the concept and operations of queues.**
- **Implement queue operations (enqueue, dequeue, peek, isEmpty).**
- **Demonstrate real-world applications of queues.**
- **Simulate queue-based systems using Python.**
- **Analyze the performance of queue operations.**

---

4. Literature Review

- **A Queue stores items in a FIFO manner. The first element added is the first one removed.**
- **Types of Queues:**
    - **Simple Queue: Basic FIFO structure**
    - **Circular Queue: Overcomes linear queue space limitations**
    - **Priority Queue: Processes elements based on priority**

- **Used in operating systems, network traffic management, and simulations.**

---

5. Methodology

1. **Implement a simple queue using a Python list.**
2. **Add methods for enqueue(), dequeue(), peek(), and isEmpty().**
3. **Simulate a real-world scenario (e.g., a ticket counter or task manager).**
4. **Display the flow of queue operations.**
5. **Optionally, visualize the queue with print() or graphical tools.**

---

6. Tools and Technologies

- **Programming Language: Python**
- **Optional: GUI using Tkinter or Matplotlib for visualization**

---

7. Expected Results

- **Efficient and orderly processing of tasks.**
- **Successful simulation of real-world queue-based systems.**
- **Clear understanding of queue operations and use-cases.**
- **Implementation of reusable code for other queue applications.**

---

8. Challenges and Solutions

Challenge	Solution
Overflow/Underflow errors	Add checks before dequeue or peek
Performance drop in large queues	Use collections.deque for optimized performance
Simulating real-world scenarios	Plan and implement step-by-step logic (e.g., task arrival and processing time)

---

9. Conclusion

This project provides a deep understanding of the queue data structure and its applications. By implementing and simulating queues, we learn how to manage and

organize tasks efficiently. Queues are a fundamental concept in many real-world systems, making this project relevant and practical.

---

10. References

1. **"Data Structures and Algorithms in Python" – Goodrich et al.**
2. **GeeksforGeeks – Queue Tutorials**
3. **Python Official Documentation – collections.deque**
4. **Real-life System Simulations – Operating Systems and Networking books**

CODE:-

Queue Class (Simple FIFO Queue)

```
class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, value):
        self.queue.append(value)
        print(f"Enqueued: {value}")

    def dequeue(self):
        if not self.is_empty():
            removed = self.queue.pop(0)
            print(f"Dequeued: {removed}")
            return removed
        else:
            print("Queue is empty!")
            return None

    def peek(self):
        if not self.is_empty():
            return self.queue[0]
```

```
        else:
            return None

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

    def display(self):
        print("Queue:", self.queue)
```

Real-World Simulation: Ticket Counter

```
import time
import random

def ticket_counter_simulation():
    q = Queue()
    customers = ["Aarush", "Athul", "Rizwan", "Nihal", "John", "Sara", "Mike"]

    print("\n--- Customers arriving at the ticket counter ---")
    for person in customers:
        q.enqueue(person)
        time.sleep(0.5)

    print("\n--- Serving customers ---")
    while not q.is_empty():
        time.sleep(1)
        q.dequeue()

ticket_counter_simulation()
```