

# Name: Tahir Fareed

# Roll No: SU92-BSAIM-F24-036

# Section: BSAI (3A)

# Subject: AI Lab

# Task : 06

Queue Implementation (With Node and Without Node)

# Introduction

A Queue is a linear data structure that follows the FIFO (First In, First Out) principle. It is similar to a real-life queue where the first person to enter is the first one to be served. Queues are widely used in scheduling, buffering, and resource management.

# Types of Queue Implementations

In this notebook, we explore two implementations of Queue:

1. Queue using Node (Linked List based implementation).

2. Queue without Node (Array/List based implementation).Queue with Node (Linked List Implementation)

In this implementation, a Queue is built using a linked list structure. Each element of the queue is represented by a Node object containing data and a pointer to the next node.

**Screenshot:**

A screen shot of a computer program

AI-generated content may be incorrect.

## Key Components:

• Node class: Represents a single element of the queue.

• Queue class: Contains 'front' and 'rear' pointers to manage the queue.

• Enqueue operation: Adds an element to the end of the queue.

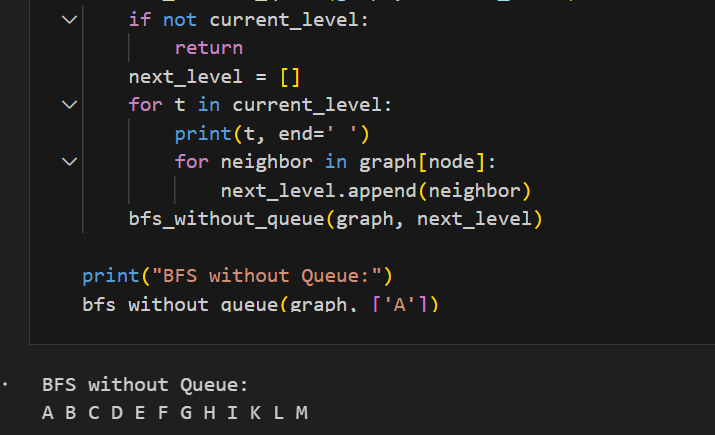
• Dequeue operation: Removes an element from the front of the queue.

• Display operation: Prints the queue elements.

# Queue without Node (Array/List Implementation)

In this implementation, a Queue is built using Python’s list (array-like structure). The front and rear indices are managed directly without the need for node pointers.

**Screenshot:**

****

## Key Components:

• List: Stores queue elements directly.

• Front index: Tracks the first element of the queue.

• Rear index: Tracks the last element of the queue.

• Enqueue operation: Adds an element at the rear of the list.

• Dequeue operation: Removes an element from the front of the list.

• Display operation: Prints the queue elements.

# Comparison

## Queue with Node:

Advantages:

• Dynamic size (no fixed capacity).

• Efficient memory usage for large, unpredictable workloads.

Disadvantages:

• Extra memory required for pointers.

• Slightly complex implementation.

## Queue without Node:

Advantages:

• Simple and easy to implement.

• Efficient for small, fixed-size queues.

Disadvantages:

• Fixed size (if implemented with arrays).

• Shifting may be required in some operations (less efficient)

# Conclusion

Both implementations of Queue are useful depending on the use case. If memory efficiency and dynamic resizing are important, linked list implementation is preferred. For simpler and small-scale problems, array/list-based implementation works well.