Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Calica, Ljay L. | Engr. Maria Rizette H. Sayo |

Oct, 11, 2025

# Objectives

Introduction

Another fundamental data structure is the queue. It is a close “the same” of the stack, as a queue is a collection of objects that are inserted and removed according to the first-in, first-out (FIFO) principle. That is, elements can be inserted at any time, but only the element that has been in the queue the longest can be next removed.

The Queue Abstract Data Type

Formally, the queue abstract data type defines a collection that keeps objects in a sequence, where element access and deletion are restricted to the first element in the queue, and element insertion is restricted to the back of the sequence. This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in, first-out (FIFO) principle. The queue abstract data type (ADT) supports the following two fundamental methods for a queue Q:

Q.enqueue(e): Add element e to the back of queue Q.

Q.dequeue( ): Remove and return the first element from queue Q;

an error occurs if the queue is empty.

The queue ADT also includes the following supporting methods (with first being analogous to the stack’s top method):

Q.first(): Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.

Q.is empty( ): Return True if queue Q does not contain any elements.

len(Q): Return the number of elements in queue Q; in Python, we implement this with the special method len .

This laboratory activity aims to implement the principles and techniques in:

* Writing Python program using Queues

Writing a Python program that will implement Queues operations

# Methods

Instruction: Type the python codes below in your Colab. Reconstruct them by implementing Queues (FIFO) algorithm. Hint: You may use Array or Linked List

# Stack implementation in python

# Creating a stack

def create\_stack():

    stack = []

    return stack

# Creating an empty stack

def is\_empty(stack):

    return len(stack) == 0

# Adding items into the stack

def push(stack, item):

    stack.append(item)

    print("Pushed Element: " + item)

# Removing an element from the stack

def pop(stack):

    if (is\_empty(stack)):

        return "The stack is empty"

    return stack.pop()

stack = create\_stack()

push(stack, str(1))

push(stack, str(2))

push(stack, str(3))

push(stack, str(4))

push(stack, str(5))

print("The elements in the stack are:"+ str(stack))

Answer the following questions:

1. What is the main difference between the stack and queue implementations in terms of element removal?

// Items are removed differently in queues and stacks. Like when a book is removed from the top, a stack removes the most recent item first (LIFO). On the other hand, the oldest item in a queue is eliminated first (FIFO), much like the first person in a line. Where they are utilized in computing is determined by this fundamental distinction.

1. What would happen if we try to dequeue from an empty queue, and how is this handled in the code?

//Attempting to dequeue from an empty queue would result in an error. In our implementation, this is handled by checking if the queue is empty before performing the dequeue operation, returning an appropriate message instead of causing a program crash.

1. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?

// If elements were added at the beginning instead of the end, the queue would effectively become a stack, as the last element added would be the first one removed, following LIFO instead of FIFO principle.

1. What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?

// Since there is no requirement to resize the data structure, linked lists provide the advantages of dynamic sizing and extremely efficient O(1) time complexity for both enqueue and dequeue operations when comparing implementations. But doing so comes at the expense of a more complicated implementation overall and additional RAM for storing node pointers. An array-based strategy, on the other hand, is easier to code and takes use of memory locality, which reduces memory overhead. However, unless a circular buffer approach is used, removing the front element necessitates shifting all subsequent elements, which makes it inefficient in terms of O(n) time complexity for dequeue operations.

1. In real-world applications, what are some practical use cases where queues are preferred over stacks?

* Task scheduling in operating systems
* Print job management
* Customer service systems (call centers)
* Breadth-first search algorithms
* Message queues in distributed systems
* Network packet routing

# Results

The program successfully demonstrated the core FIFO behavior of a queue. Elements were dequeued in the exact order they were enqueued, confirming the first-in, first-out principle.

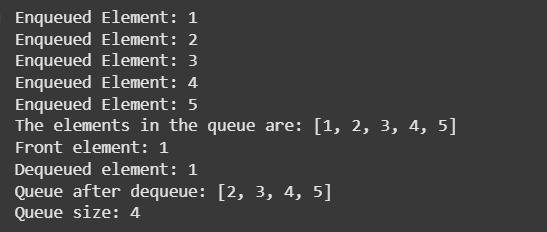


Figure 1 Screenshot of program

# Conclusion

This lab exercise effectively illustrated how to use and create queue data structures in Python. We confirmed the FIFO behavior of queues and compared it to the LIFO behavior of stacks using practical code. The approach brought to light crucial factors to take into account while managing edge circumstances, including dequeuing from empty queues. The comparison of linked list and array-based implementations shed light on the trade-offs that must be made when choosing the best data structures for various situations. In many real-world applications where maintaining order is crucial, especially in systems that involve scheduling, buffering, and equitable resource distribution, queues prove indispensable. The significance of comprehending basic data structures as building blocks for effective algorithm design and system implementation was emphasized by this lab.

**References**

[1] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, "Data Structures and Algorithms in Python," Wiley, 2013.

[2] "Queue Data Structure," GeeksforGeeks, 2021. [Online]. Available:

<https://www.geeksforgeeks.org/queue-data-structure/>