UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Delinia, Filjohn B.

Instructor:
Engr. Maria Rizette H. Sayo

October 11, 2025

I. 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

II. 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?
2. What would happen if we try to dequeue from an empty queue, and how is this handled in the code?
3. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?
4. What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?
5. In real-world applications, what are some practical use cases where queues are preferred over stacks?

III. Results

```python
# Queue implementation in python

def create_queue():
    queue = []
    return queue

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

def enqueue(queue, item):
    queue.append(item)
    print("Enqueued Element: " + item)

def dequeue(queue):
    if is_empty(queue):
        return "The queue is empty"
    return queue.pop(0)


queue = create_queue()
enqueue(queue, str(1))
enqueue(queue, str(2))
enqueue(queue, str(3))
enqueue(queue, str(4))
enqueue(queue, str(5))

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

print("Dequeued Element: " + dequeue(queue))
print("The elements in the queue after dequeue:" + str(queue))
```

Figure 1. Source Code

```
Enqueued Element: 1
Enqueued Element: 2
Enqueued Element: 3
Enqueued Element: 4
Enqueued Element: 5
The elements in the queue are:['1', '2', '3', '4', '5']
Dequeued Element: 1
The elements in the queue after dequeue:['2', '3', '4', '5']
```

Figure 2. Output

ANSWERS:

Q1: The main difference between the **stack** and **queue** implementations in terms of element removal, Stack removes the last added element (**LIFO**) while Queue removes the first added element (**FIFO**).

Q2: If we try to dequeue from an empty queue, the function will return "The queue is empty". The check for emptiness is handled by the is_empty() function before attempting to pop an item.

Q3: The queue would behave like a stack (**LIFO**), as new elements would be removed in reverse order.

Q4: The advantages and disadvantages of implementing a queue using linked lists versus arrays, **Linked lists** have dynamic size, efficient insert/remove at ends while an **Array** have fixed size, inefficient removal from the front.

Q5: **Call Centers** - Incoming calls are placed in a queue, and the system answers them in the order they were received, providing a fair and organized response system.

Operating Systems - Queues manage processes waiting to be executed by the CPU (CPU scheduling) and handle shared resources like printers.

Traffic Management - Queues help manage the flow of vehicles at toll booths, intersections, and other congested areas to minimize delays.

IV. Conclusion

Queues play a crucial role in programming and daily technology by ensuring that tasks are executed in the appropriate sequence—with the first request receiving priority. Whether it's coordinating data packets within networks, processing requests on websites, or arranging print jobs, queues facilitate the smooth and equitable operation of various systems. Understanding the mechanics of queues and their applications allows us to create efficient and dependable systems.

References

[1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.

[2] CCBP (2021, October 5). *Application of queue in data structure*. CCBP. https://www.ccbp.in/blog/articles/application-of-queue-in-data-structure#:~:text=real%2Dworld%20scenarios:-,1.,Data%20Buffers%20in%20Networking