UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Talagtag Mark Angel T.

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?

- In a stack, the last element that was added (pushed) is the first one to be removed (popped). This means you remove elements in the reverse order of their addition. While In a queue, the first element that was added is the first one to be removed. This follows the order in which elements were inserted, similar to a line where the person who joins first leaves first.

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

- If you try to **dequeue** (remove an element) from an **empty queue**, it typically results in an error or an exception, depending on how the queue is implemented.

3 If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?
- If the enqueue operation has been changed to add elements at the beginning of the queue, the behavior changes from First In, First Out (FIFO) to Last In, First Out (LIFO). You would remove the most recent member to be added, not the first, to effectively make the queue behave like a stack. This order reversal violates the standard queue logic, which says that the first element added should be the first one discarded.

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

- Advantages:

- can grow or shrink without a fixed limit.
- Enqueue and dequeue operations can be done in constant time O(1) if you keep track of both front and rear pointers.

Disadvantages:

- Requires extra memory for storing pointers in each node.
- Slightly more complex to implement and manage due to pointer manipulation.

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

- **Task Scheduling:** Operating systems use queues to manage processes waiting for CPU time, making sure the first task that arrives gets processed first.
- **Print Spooling:** Print jobs are lined up in a queue so documents get printed in the exact order they were sent.
- **Customer Service Systems:** Call centers and help desks handle customer requests using queues to make sure everyone is helped in the order they reached out, keeping things fair.
- **Graphs and BFS:** When searching through graphs, breadth-first search (BFS) uses a queue to explore nodes level by level, visiting neighbors before diving deeper.
- **Order Processing:** Whether it's online shopping or manufacturing, orders are handled in the order they come in to keep everything organized and fair.

III. Results

```
Pushed Element: 1
Pushed Element: 2
Pushed Element: 3
Pushed Element: 4
Pushed Element: 5
The elements in the stack are:['1', '2', '3', '4', '5']
```

IV. Conclusion

In summary, stacks and queues are fundamental data structures with distinct behaviors based on how elements are added and removed. A stack follows a LIFO (Last In, First Out) approach, where the most recently added element is the first to be removed, while a queue follows a FIFO (First In, First Out) approach, where the first element added is the first to be removed.

References

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