UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
AMPONG, J-KEVIN L.

Instructor:
Engr. Maria Rizette H. Sayo

10, 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

CODE RESULT SNIPPET:

```
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']
```

1. Stacks and queues differ mainly in their removal order. Stacks are LIFO (Last-In, First-Out), meaning the last item added is the first one removed (from the "top"). Queues are FIFO (First-In, First-Out), meaning the first item added is the first one removed (from the "front").

2. Attempting to remove an item from an empty queue (dequeue) usually results in an error or exception. In contrast, the specific code mentioned handles an empty stack by returning a message ("The stack is empty") instead of causing an error.

3. If you change a queue's "enqueue" operation to add new items to the *beginning* (front) instead of the end, you reverse its behavior. This transforms the queue into a stack, as the most recently added item would then be the first one removed, following the LIFO principle instead of FIFO.

4. The provided lists of advantages and disadvantages for queue implementation appear contradictory. Key potential advantages include a dynamic size and the fundamental enqueue (add) and dequeue (remove) operations. However, potential disadvantages can include inefficient removal (if items must be shifted), a fixed size (in some implementations), or higher memory use.

5. Queues are used in applications where the order of arrival is critical. For example, they manage client requests to ensure the first request received is the first one processed, preventing later requests from unfairly skipping ahead.

IV. Conclusion

In this laboratory exercise, we learned about the Queue, a basic computer science structure. It operates like a waiting line: the first item to enter is the first item to leave (FIFO). The queue's essential actions are to add new items to the end and take items out from the beginning.

References

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