UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Catahan, Joshua A.

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

Reconstructed code:

```
# Creating a queue
def create_queue():
    queue = []
    return queue


# Check if the queue is empty
def is_empty(queue):
    return len(queue) == 0
```

```
# Adding items into the queue (Enqueue)
def enqueue(queue, item):
    queue.append(item)
    print("Enqueued Element: " + item)

# Removing an element from the queue (Dequeue)
def dequeue(queue):
    if is_empty(queue):
        return "The queue is empty"
    return queue.pop(0)

# Main program
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))

# Demonstrate FIFO removal
print("Dequeued element:", dequeue(queue))
print("Queue after one dequeue:", queue)
```

Answers:

1. The main difference is that a stack removes the most recently added element, while a queue removes the earliest one added.
2. When we try to dequeue from an empty queue, the program shows a message saying it's empty to prevent an error.
3. If elements are added at the beginning instead of the end, the queue will no longer follow the FIFO rule and will behave like a stack.
4. A linked list allows dynamic memory use and is easier to expand, while an array is faster for access but has a fixed size.
5. Queues are often used in things like printer task management, online order processing, or customer service systems where tasks must follow a first-come, first-served process.

IV. Conclusion

Through this activity, I understood how a queue works using the FIFO principle and how it differs from a stack. I also learned how to apply queue operations such as enqueue and dequeue in Python. This helped me see how data structures control the flow of information in programs. Queues are simple but very useful, especially in systems where order and timing matter, like handling requests, managing tasks, or scheduling processes.

References

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