UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Barbas, Steven Jade P.

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

```
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)

def front(queue):
    if (is_empty(queue)):
        return "The queue is empty"
    return queue[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("\nDequeuing elements:")
print("Dequeued Element: " + dequeue(queue))
print("Dequeued Element: " + dequeue(queue))

print("\nThe remaining elements in the queue are: "+ str(queue))
```

Figure 1 Screenshot of program

```
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']

Dequeuing elements:
Dequeued Element: 1
Dequeued Element: 2

The remaining elements in the queue are: ['3', '4', '5']
```

Figure 2 Screenshot of output

Answers:

1, The main difference between the stack and queue implementation in terms of element removal is in a stack, the last item you added is the first one to come out. In a queue, the first item you added is the first one to come out.

2. If you try to dequeue from an empty queue, the code checks if the queue is empty first. If it is empty, instead of causing an error, it returns the message "The queue is empty". This prevents the program from crashing and tells you why the code didn't work.

3. If we modify enqueue to add elements at the beginning instead of the end, it would completely break the queue's normal behavior. Instead of being FIFO (First-In-First-Out), it would become LIFO (Last-In-First-Out), just like a stack.

4. Linked list queues can more data without limits and are fast, but use more memory and are harder to code. Array queues are simple to make and memory-friendly, but have limited

size and get slow when removing items. Choose linked lists for flexibility or arrays for simplicity.

5. In a fast food restaurant , using queues ensure the first customer in line gets served first. Using a stack would be unfair the last person would order first. Queues maintain proper order and fairness in customer service.

IV. Conclusion

In conclusion, In this activity I was able to reconstruct the stack implementation (LIFO) in python that was given to Queues implementation (FIFO) in python. Queues are already tackled in our previous lessons that why I was able to execute it. Queues and stack apply in real world making them essential for maintaining order and fairness in real-world applications like food service, printing, and customer support, where maintaining order and fairness matters most.

References

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