UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Uy, Junichiro H.

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
```

```python
# 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

1. What is the main difference between the stack and queue implementations in terms of element removal?

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

They are different in the sense that, queues remove the elements that was first added, while stacks remove the elements that was recently added. So queues use pop(0) while stack use pop().

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

```python
if (is_empty(queue)):
        return "The queue is empty"
```

The code would return “The queue is empty” and won’t be able to dequeue anything.

3. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?

```
anada_queue = create_queue()

def enqueue_front(queue, item):
    queue.insert(0, item)
    print(f"Added Element: {item}")

enqueue_front(anada_queue, 1)
enqueue_front(anada_queue, 2)
enqueue_front(anada_queue, 3)

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

Output:

```
Added Element: 1
Added Element: 2
Added Element: 3
The elements in the queue are:[3, 2, 1]
```

If we were to do that, it would become like a stack, it would add at the very first and would remove the very first, so it would be like first-in last-out. This would make it a stack, not a queue.

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

Array:

Advantage: It's very easy to implement, you just need to make a list and add functions that resembles queues functions.

Disadvantage: Removing the first element [0] is very slow because all the elements after it has to fill the gap, so everything has to move.

Linked List:

Advantage: Enqueuing and Dequeuing are both fast because you only have to move and change pointers of the head and tail.

Disadvantage: It uses a bit more memory because it has to store pointers for each element.

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

- Queues are helpful for managing and showing first in first our system, so it is commonly used in restaurants to prioritize who ordered first. It can also be used for traffic lights. First to turn green, would be the first to turn red, and vice versa.

IV. Conclusion

In short, Queues is a useful data type. It can be used for different applications that require it, usually those that needs automated storage system, delete system, and more. This laboratory demonstrated how to use queues for arrays, using enqueues, dequeues, and checking if it’s empty.

References

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