UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Poliño, Justine

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

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

While pop(0) is used to remove entries from the beginning of queues (FIFO), pop() is used to remove elements from the end of stacks (LIFO).

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

Instead of generating an error if we try to dequeue from an empty queue, the dequeue() function utilizes is_empty(queue) to check if the queue is empty before returning the message "The queue is empty".

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

If we modified the enqueue to add elements at the beginning (using insert(0, item) instead of append(item)), FIFO behavior would still be preserved even if the code would be different. However, this would be less effective for arrays because inserting at the beginning requires shifting every element.

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

- Linked List Advantages:
  - **Faster Operations:** Both of them are adding (enqueue) and removing (dequeue) take constant time (O(1)).
  - **Dynamic Size:** Grows and shrinks automatically as needed.
- **Linked List Disadvantages:**
  - **More Memory:** Uses extra memory for storing pointers.
  - **Slower Access:** Poor cache performance due to non-contiguous memory.
- **Array Advantages:**
  - **Memory Efficient:** Only stores data, no extra pointers.
  - **Faster Access:** Better cache performance due to contiguous memory.
- **Array Disadvantages:**
  - **Slower Removal:** removing from the front is slow (O(n)) due to element shifting.
  - **Fixed Size:** Basic arrays have limited capacity (though dynamic arrays can resize).

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

Queues are used for **fair, ordered processing** where "first in, first out" matters:

- **CPU/Process Scheduling** - OS manages processes in order
- **Print Spooling** - Print jobs processed in received order
- **Message Systems** - Messages handled sequentially (RabbitMQ, SQS)
- **Web Servers** - HTTP requests processed in arrival order
- **Breadth-First Search** - Graph nodes explored level by level
- **Call Centers** - Customers served in calling order

Figure 1 Screenshot of program

IV. Conclusion

I successfully converted a stakc to queue by changing the removal method . Arrays works on simple cases, but linked lists are more efficient . Queues are needed for ordered processing in real world systems like task scheduling and messaging w

References

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