UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Asugas, Kenneth R.

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?

In a stack, elements are removed from the top using LIFO (Last-In, First-Out). In a queue, elements are removed from the front using FIFO (First-In, First-Out). This means the element inserted earliest in a queue is the first to be removed and the latest in stack is the first to be removed.

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

Trying to dequeue from an empty queue will cause an error or invalid operation. In the code, this is handled by checking if the queue is empty using is empty(queue) before performing the pop(0) operation.

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

If this were done, the removal order would be in the opposite direction, and it would mimic a stack (LIFO) rather than a queue (FIFO). The last one that was inserted would be removed first.

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

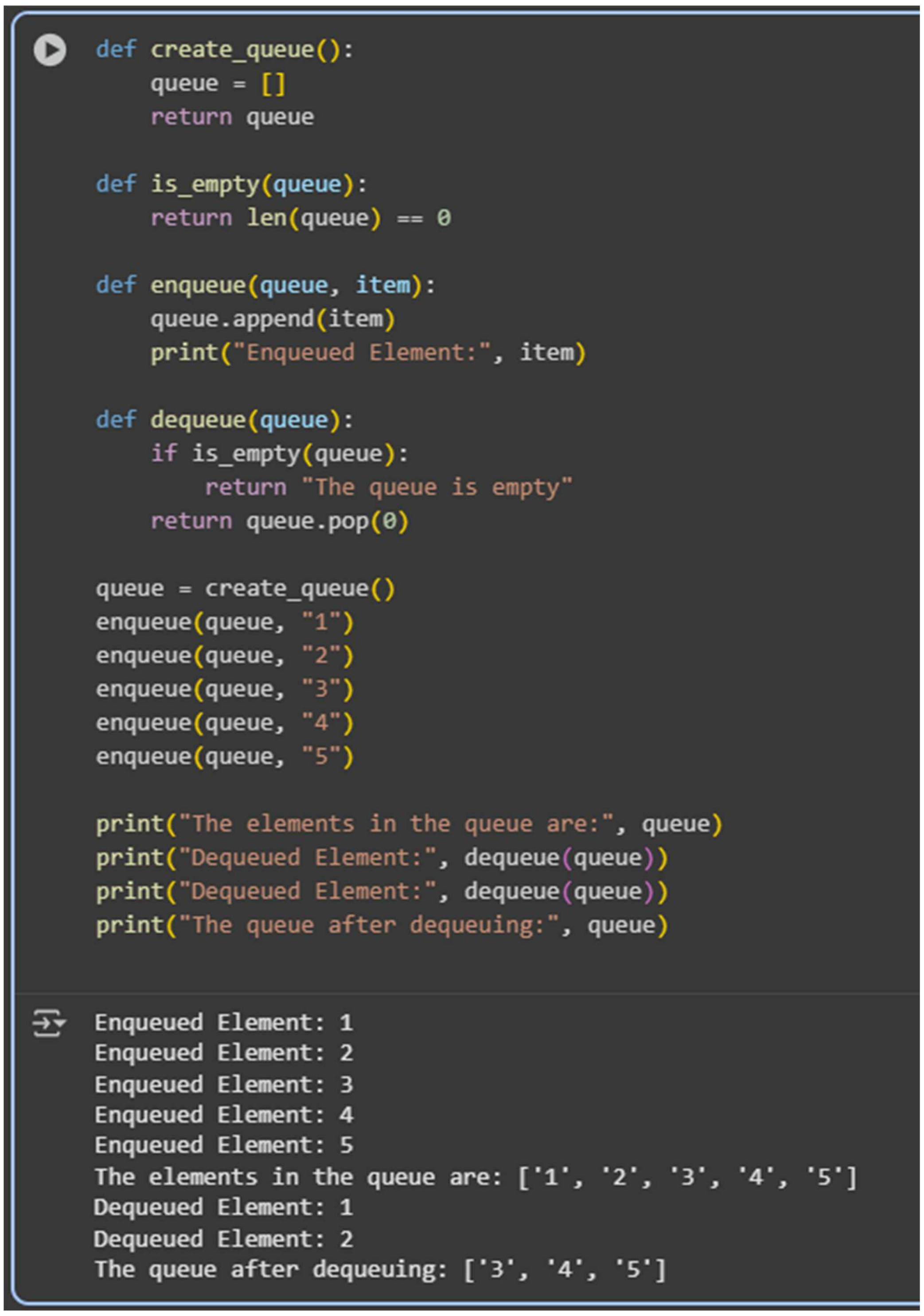
Linked List:
Advantages: Dynamic size, low memory usage, quick insertion and deletion.
Disadvantages: Additional memory for pointers, slower access to a certain element.

Array:
Advantages: Easier to implement, quicker element access.
Disadvantages: Fixed size, inefficient when numerous dequeue operations lead to shifting elements.

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

Queues are used in real-world applications in CPU task scheduling, resource allocation, batch processing, Batch Processing, printer job management, network data buffering, customer service systems, and breadth-first search (BFS) in graph algorithms, where the first request or data received must be handled first.

III. Results



```python
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)

queue = create_queue()
enqueue(queue, "1")
enqueue(queue, "2")
enqueue(queue, "3")
enqueue(queue, "4")
enqueue(queue, "5")

print("The elements in the queue are:", queue)
print("Dequeued Element:", dequeue(queue))
print("Dequeued Element:", dequeue(queue))
print("The queue after dequeuing:", queue)
```

```
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']
Dequeued Element: 1
Dequeued Element: 2
The queue after dequeuing: ['3', '4', '5']
```

Figure 1 Screenshot of program

The output shows that the program properly implements the queue's First-In, First-Out (FIFO) property. Every element between 1 and 5 is added one by one to the queue, As observed at the enqueued element lines. When de-queueing two elements, the first two elements inserted (1 and 2) are de-queued first, resulting in ['3', '4', '5'] still remaining in the queue. This ensures that the queue handles elements precisely the same order they have been inserted with correct sequential processing of data.

IV. Conclusion

In this lab report, I successfully implemented and exhibited the Queue (FIFO) data structure in Python. The exercise emphasized the difference between queues and stacks, especially the way elements are pushed and popped. Queues are particularly suited to tasks involving ordered processing of data. Through this experiment, we achieved a better understanding of how Queue operations could be used to solve real-life problems like scheduling and buffering.

References

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