UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
LastName, FirstName MI.

Instructor:
Engr. Maria Rizette H. Sayo

Month, DD, YYYY

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 imple  mentation 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))
```

Python Program implementing Queue:

```
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 peek(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("Dequeued Element: " + dequeue(queue))
print("Queue after one dequeue: " + str(queue))
```

Output of the 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']
Dequeued Element: 1
Queue after one dequeue: ['2', '3', '4', '5']
```

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

Answers:

1. The main difference is *where* elements are removed. In a stack, we remove the last item that was added (LIFO or last in, first out) In a queue, we remove the first item that was added (FIFO or First in, last out).
2. If we try to dequeue from an empty queue, It will have an outpu of “The queue is empty” because there’s nothing to remove. It’s like a safety check to avoid errors.
3. If we change enqueue to add things at the start instead of the end, it would mess up the queue idea. It’d start acting like a stack since the newest item would get removed first.
4. Using linked lists is good because you can easily add or remove elements without worrying about size limits, but it takes a bit more memory since you need to store pointers. Arrays are simpler and faster for accessing elements, but they have a fixed size (unless you resize them manually), which can be a hassle.
5. Queues are used a lot when order matters, like printing documents (printing in the order that it was sent) customer service lines, or task scheduling in operating systems. Basically anywhere something should happen “first come, first served.”

IV. Conclusion

In conclusion, the code clearly shows how stacks and queues work in opposite ways when it comes to adding and removing elements. The stack follows a last-in, first-out setup, while the queue uses first-in, first-out. It also shows how basic checks like “is empty” prevent errors when removing items. Overall, it’s a good example of how data organization affects how a program handles information like whether something gets processed first or last.

References

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