UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Sorellano, John Kenneth T.

Instructor:
Engr. Maria Rizette H. Sayo

October, 11, 2005

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?

- The main difference between stack and queue implementation are stacks is (LIFO) or Last in, first out which means the last element you will add will be the first one to be popped out, while the queue is (FIFO) or First in, First out which means the first element you will add will be the first one to be popped out.

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

- If we try to dequeue from an empty list it will cause an error to 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?

- The queue will behave like a stack implementation because we add elements in the beginning of the list instead of the end.

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

- Arrays are fast and easy, but they take more space while linked lists allow queues to expand and handle additions and deletions quickly, but they also use more memory.

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

- In real life applications, some of practical use of queues are Customer service support, Ordering system, Call waiting system because it shows (FIFO) function.

III. Results

```
def create_queue():
    return []

def enqueue(queue, element):
    queue.append(element)

def dequeue(queue):
    if len(queue) == 0:
        return "Empty Queue"
    return queue.pop(0)

def peek(queue):
    if len(queue) == 0:
        return "Empty queue"
    return queue[0]

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

print("In Queue:", num)
print("Dequeued:", dequeue(num))
print("In Front:", peek(num))
print("Latest queue:", num)
```

```
In Queue: ['1', '2', '3', '4', '5']
Dequeued: 1
In Front: 2
Latest queue: ['2', '3', '4', '5']
```

IV. Conclusion

In Conclusion, This activity shows how stacks and queues function differently and how i turned the stack source code to a queue implementation.

References

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