Namal University Mianwali

Department of Electrical Engineering

Data Structures and Algorithms

Course Title: CSC-200L

Lab # 7 Manual

Implement Queue and Deque Abstract Data Types with Array-Based Data Structure

Name	
Roll No.	

Instructor: Dr. Farukh Qureshi

Lab Engineer: Engr. Sana Perveen

Version1.0	**Engr. Naureen Shaukat**	10/03/2023
Version1.1	**Engr. Misbah Batool**	28/02/2024
Version2.0	**Engr. Sana Perveen**	7/11/2025

In previous lab, we studied the concept of stacks and their implementation in python. Stack works on the process of Last in First Out (LIFO).

1. Lab Objectives

The objective of this lab is to introduce students to the concept of Queues and DEque

2. Lab Outcomes

CLO 1: Implement fundamental Data Structure using Python Programming Language.

CLO-2: Analyze computation cost of different algorithms.

3. Equipment

Software

- Spyder

4. Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
3. The report should be submitted on the given template, including:
    a. Code (copy and pasted, NOT a screenshot)
    b. Output figure (as instructed in 3)
    c. Explanation where required
4. The report should be properly formatted, with easy-to-read code and easy to see figures.
5. Plagiarism or any hint thereof will be dealt with strictly. Any incident where plagiarism is caught, both (or all) students involved will be given zero marks, regardless of who copied whom. Multiple such incidents will result in disciplinary action being taken.

5. Introduction

5.1 Queue:

As stack works on the concept of Last in First Out (LIFO), Queue is a linear data structure that works on the principle of First In First Out (FIFO). Elements can be inserted at any time from one end and the elements that have been longest in the queue will be removed first.
Real-time application of Queue can be passengers standing at the counter of a bus station to purchase tickets.

Graphical Representation of Queue is given below in Figure 1.

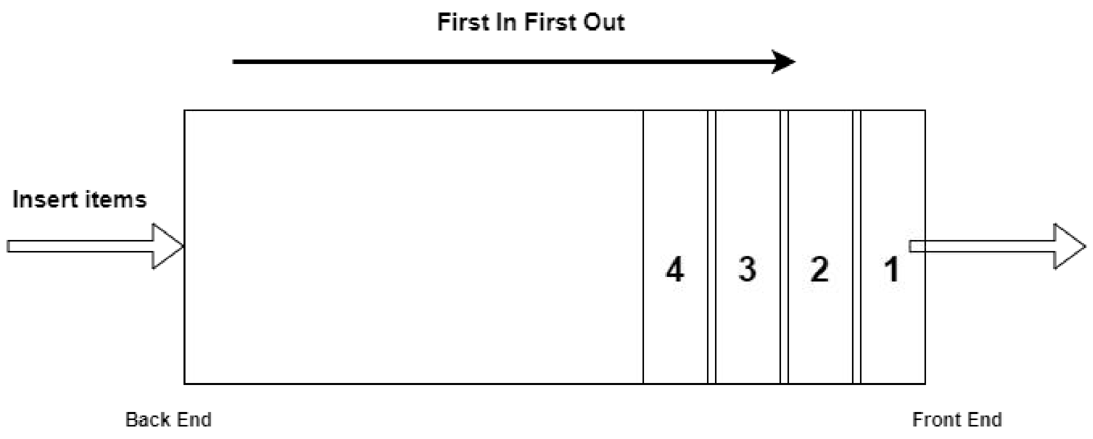


Figure 1 Graphical Representation of Queue

There are two processes being conducted inside Queue,

a) Enqueue: Add elements to the back of the queue.
b) Dequeue: Remove elements from the front of the Queue.

5.2 Implementation of Queue in Python:

The components required for Implementing queue through circular array requires following components.

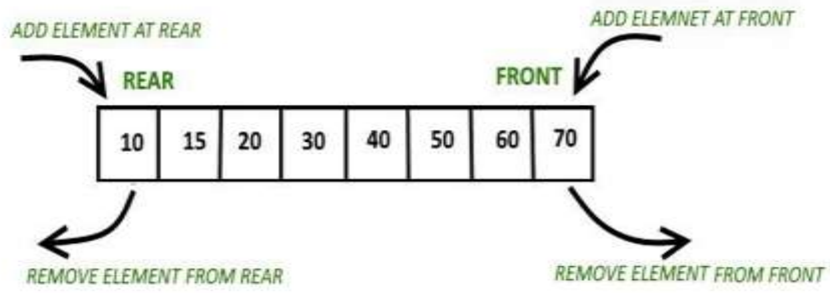
1. Class

a) __init__ method
b) Method for length
c) Method that checks if queue is empty
d) Method that returns the element of queue at front
e) dequeue method
f) enqueue method
g) resize method
h) print the whole queue

2. Main

Double Ended Queue:

Data structure that supports insertion and deletion at both the front and the back of the queue. Such a structure is called a double ended queue, or deque, which is usually pronounced “deck” to avoid confusion with the dequeue method of the regular queue ADT, which is pronounced like the abbreviation “D.Q. Mainly the following four basic operations are performed on deque: **insertFront():** Adds an item at the front of Deque. **insertRear():** Adds an item at the rear of Deque. **deleteFront():** Deletes an item from front of Deque. **deleteRear():** Deletes an item from rear of Deque.



The deque abstract data type is more general than both the stack and the queue ADTs. The extra generality can be useful in some applications. For example, we described a restaurant using a queue to maintain a waitlist. Occasionally, the first person might be removed from the queue only to find that a table was not available; typically, the restaurant will re-insert the person at the first position in the queue. It may also be that a customer at the end of the queue may grow impatient and leave the restaurant. Here is a basic example of double ended queue in Table1. In this example, add_first method is used to add elements at the front of deque. Add_last() method is used to add elements at the last of deque. Delete_last() method is used to delete elements at the last of deque.

Operation	Return Value	Deque
D.add_last(5)	–	[5]
D.add_first(3)	–	[3, 5]
D.add_first(7)	–	[7, 3, 5]
D.first()	7	[7, 3, 5]
D.delete_last()	5	[7, 3]
len(D)	2	[7, 3]
D.delete_last()	3	[7]
D.delete_last()	7	[ ]
D.add_first(6)	–	[6]
D.last()	6	[6]
D.add_first(8)	–	[8, 6]
D.is_empty()	False	[8, 6]
D.last()	6	[8, 6]

Python Code for QUEUE ADT:

```
class Queue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.items = []

    def length(self):
        return len(self.items)

    def is_empty(self):
        return len(self.items) == 0

    def is_full(self):
        return len(self.items) == self.capacity

    def front(self):
        if self.is_empty():
            return None
        return self.items[0]

    def enqueue(self, item):
        if self.is_full():
            raise IndexError("Queue is full")
        self.items.append(item)
    def resize(self, new_size):
        if new_size < self.length():
            self.items = self.items[:new_size]
        self.capacity = new_size
```

def print(self): # print the queue

# Students are required to design this method by themselves.

# Make code for the main

* print commands in the above example is added to make your understanding better about the code.

LabTasks

Task 1:

a) Make DEQUEUE method for the class provided in Example.

b) Design PRINT method for printing the whole queue in Example.

c) Write the main portion for making a queue. Call all the methods to see the behavior of the code.

d) State the time complexity of enqueue and dequeue operations in your array-based queue, and briefly explain why.

Python Code for DEQUE ADT:

```
class Deque:
    def __init__(self, capacity):
        self.capacity = capacity             # Maximum size of deque
        self.arr = [None] * capacity         # Circular array
        self.front = -1                      #Points to front element
        self.rear = -1                       #Points to rear element
        self.size = 0                        #Number of elements

    def is_empty(self):
        return self.size == 0

    def is_full(self):
        return self.size == self.capacity

    def insert_front(self, value):
        if self.is_full():
            print("Deque is FULL. Cannot insert at front.")
            return False

        # If empty, both pointers go to 0
        if self.front == -1:
            self.front = 0
            self.rear = 0
        else:
            self.front = (self.front - 1 + self.capacity) %
self.capacity

        self.arr[self.front] = value
        self.size += 1
        return True

    def delete_front(self):
        if self.is_empty():
            print("Deque is EMPTY. Cannot delete from front.")
            return False

        # If there's only one element
        if self.front == self.rear:
            self.front = -1
            self.rear = -1
        else:
            self.front = (self.front + 1) % self.capacity

        self.size -= 1
        return True

    def get_front(self):
        if self.is_empty():
            return None
        return self.arr[self.front]

    def get_rear(self):
        if self.is_empty():
            return None
        return self.arr[self.rear]
```

Task 2:
Using the above example code, make a method (insert_back) to insert elements at the back side of the deque. Also, make a method (delete_back) to delete elements at the back side of the deque.
Make a method (print_deque) to print the deque. Also, make a main method to call the above methods for output.
Also state the time complexity of the operations insert_front, delete_front, insert_back and delete_back, and briefly explain the reason based on how elements are stored and accessed in the array. Also explain what happens to the time complexity if removing an element requires shifting the remaining elements.

Task 3:
Write a python program to implement a ticket booking system where customers can book tickets either from the front or the back of the double ended queue. The system should have the following methods:

book_front(ticket): Adds a ticket to the front of the queue.
book_back(ticket): Adds a ticket to the back of the queue.
cancel_front(): Cancels the ticket at the front of the queue.
cancel_back(): Cancels the ticket at the back of the queue.
get_total_tickets(): Returns the total number of tickets booked.

Make a class TicketBookingSystem to implement the deque. Also make a main method to call the above methods for output (Number of seats booked and canceled from front and back, and total number of seats booked.). You can import deque using this piece of code: from collections import deque.

Evaluation Rubric

CLO 1: Implement fundamental Data Structure using Python Programming Language

CLO-2: Analyze computation cost of different algorithms.

Total Lab Weightage: 3.57%

Criteria	Excellent (10)	Good (8-9)	Satisfactory (6-7)	Improvement (4-5)	Satisfactory (0-3)	Obtained Marks
Understanding	Demonstrates a deep understanding of fundamental Python concepts, syntax, and principles. Able to explain the rationale behind code choices comprehensively.	Shows a solid understanding of Python fundamentals, with clear explanations of code decisions.	Grasps fundamental Python concepts; explanations may lack depth or clarity in some areas.	Demonstrates basic understanding; explanations may be limited or unclear.	Lacks understanding; unable to explain code choices effectively.	
Code Quality	Code is exceptionally well-structured, follows best practices, and is highly readable. Efficient use of Python idioms and constructs.	Code is well-organized, readable, and follows best practices. Effective use of Python idioms.	Code is mostly clear and follows acceptable practices, though some areas may lack clarity or optimal structure.	Code readability is a concern, inconsistent organization or structure.	Unsatisfactory written code; difficult to read or understand. Frequent violations of best practices.	
Report Writing	Exceptional report writing skills; clear, concise, and well-organized. No grammar or formatting mistakes.	Report is well-written with. Minor grammar or formatting mistakes.	Adequate report; Noticeable grammar and formatting mistakes.	Basic report; Frequent grammar or formatting issues.	Unsatisfactory report. Severe grammar and formatting problems.	