Namal University, Mianwali
Department of Electrical Engineering
Course Title: EE-253 Data Structures and Algorithms

<u>LAB MANUAL</u>

Lab 8 Deques in Python

V1: Designed by: Engr. Ali Hasnain

In previous lab, we have studied the concept of queue in python. We have also studied how to implement a queue using array and performed different operations (enqueue, dequeue, front, back, etc.) on queue.

1. Lab Objectives

The objective of this lab is to introduce students to the concept of double ended queue (deques) in python. In this lab, the students will enable the concepts of implementing a deque, accessing the elements and operations which can be performed on deques. Students will be provided with examples, followed by performing lab tasks.

2. Lab Outcomes

- CLO:1 Recognize the usage of fundamental Data structure using Python Programming Language.
- CLO:3 Demonstrate solution to real life problems using appropriate data structures.

3. Equipment

- Software
  - IDLE (Python 3.11)

4. Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as 'Example') are for practice purposes only while others (marked as 'Task') have to be answered in the report.
3. 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.
4. 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
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. 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.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

5. Background

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.

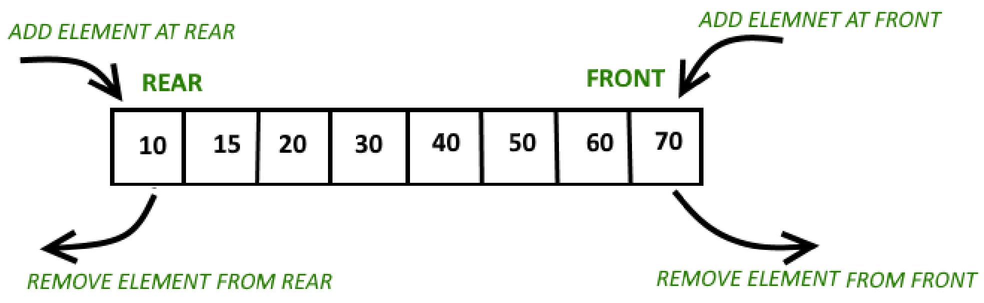
Mainly the following four basic operations are performed on queue:

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 dque. 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]

Figure 1 Table1: Example of Deque

Example 1: Below, we present an example of the use of our Double_Ended_Queue class to implement double ended queue. In this example we have implemented different methods (e.g. get_front, insert_front, delete_front, etc.) of double ended queue.

Source code:

```
class Double_Ended_Queue:
    def __init__(self, capacity):
        """Initialize a Deque object with a fixed capacity"""
        self.capacity = capacity
        self.front = -1     # Pointer to the front element
        self.back = 0       # Pointer to the rear element
        self.size = 0       # Current number of elements in the deque
        self.arr = [None] * self.capacity   # Circular array to hold the elements

    def is_empty(self):
        """Check if the deque is empty"""
        return self.size == 0

    def get_front(self):
        """Get the value of the front element"""
        if self.is_empty():
            return None
        return self.arr[self.front]

    def insert_front(self, value):
        """Insert an element at the front of the deque"""
        if self.front == -1:    # If deque is empty
            self.front = 0
            self.back = 0
        elif self.front == 0:   # If front pointer is at the start of the array
            self.front = self.capacity - 1   # Set front pointer to the end of the array
        else:
            self.front -= 1   # Decrement front pointer to insert element at the new front position
        self.arr[self.front] = value   # Insert the value at the new front position
        self.size += 1   # Increment the size of the deque
        return True

    def delete_front(self):
        """Delete the element at the front of the deque"""
        if self.is_empty():
            return False   # Return False if deque is already empty
        if self.front == self.back:   # If there's only one element in the deque
            self.front = -1
            self.back = -1
        elif self.front == self.capacity - 1:   # If front pointer is at the end of the array
            self.front = 0   # Set front pointer to the start of the array
        else:
            self.front += 1   # Increment front pointer to delete the current front element
        self.size -= 1   # Decrement the size of the deque
        return True

# Create a deque with a capacity of 4
d = Double_Ended_Queue(4)
d.insert_front(2)
d.insert_front(1)
print("Front element:", d.get_front())
print("Delete from front",d.delete_front())
print("Is list empty:",d.is_empty())
```

<u>Output:</u>

```
Front element: 1
Delete from front True
Is list empty: False
```

6. Lab Tasks:

Task 1: Using above example code, make a method (insert_back) to insert elements at the back side of deque. Also make a method (delete_back) to delete elements at the back side of deque.

Task 2: Make a method (print_deque) to print above Task 1. Also make a main method to call the above methods for output.

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

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-2**	**Marginal 3-5**	**Satisfactory 6-8**	**Exemplary 9-10**	**Allocated Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested byothers. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend forthe instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around. More than two incidents of talking non-lab related stuff in laband/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff in lab Voice level exceeding theappropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceedingthe appropriate level, use of cell phones and involvementin any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Tookcare of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	