National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

CS250: Data Structures and Algorithm

Class: BS2k21_Mathematics – Fall 2023

Lab 08:

Stacks and Queues

Linear queues, Circular Queues and Deques

Date: 1st November, 2023

Time: 10:00 am - 01:00 pm

Instructor: Fauzia Ehsan

Submission Deadline: After MSE

National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

Lab 08: Implementation of Stacks and Queues in different problems

Introduction

This lab is based on stacks and queues and its implementation statically and dynamically(via deques).

Objectives

Objective of this lab is to get familiar with the queues ADT and its implementation in a programming language. Queue data structure has many applications in computing such as holding files awaiting printer access.

Tools/Software Requirement

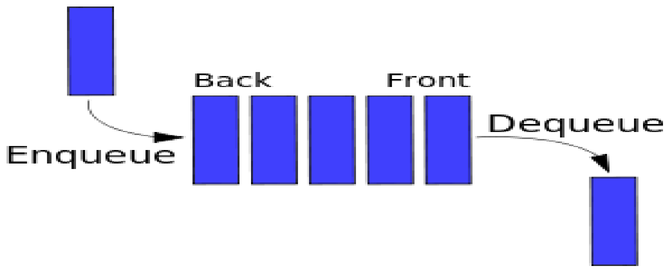
Python 3/ PyCharm IDE / VS Code

Contents

Description:

Queues:

In computer science, a queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position and removal of entities from the front terminal position. This makes the queue a First-In-First-Out (FIFO) data structure.

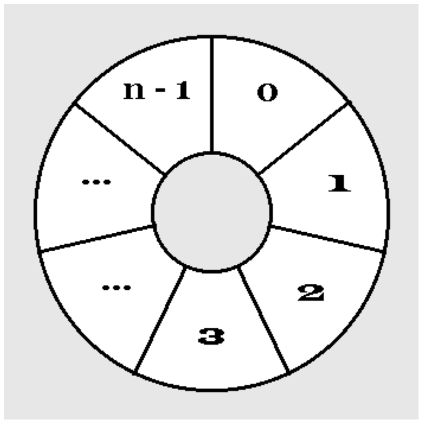


The following sets of operation are generally supported by queue:

1. **`void Enqueue(element) - add an element at the rear end of the queue`**
2. **`element Dequeue() - removes and display the element from the front end of the queue`**
3. **`bool isEmpty() - checks if the queue is empty or not`**
4. **`bool isFull() - checks if the queue is full or not`**
5. **`void Clear() - release the memory allocated by queue`**
6. **`void Peek()` – display the contents of first element of queue at front location**

"Circular" Array implementation of a Queue

We can think of the array of elements as a circular list by "gluing" the end of the array elements[0..n-1] to its front as in the following picture.



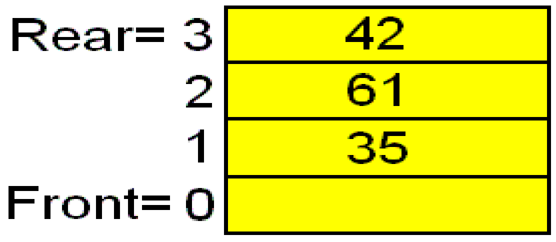
In this situation, the queue indexes front and rear advance by moving them clockwise around the array. To achieve this, addition modulo n is used in the enqueue and dequeue operations described above. Thus, when **rear = n - 1**, the end of the array, if there is room, *rear* is incremented by 1 as follows:

rear = (rear + 1) % n = (n - 1 + 1) % n = n % n = 0

A similar formula is used for front when dequeuing.

The only difficulty with this scheme involves detecting the queue-empty and queue-full conditions. It seems reasonable to select as the queue-empty condition *front is one slot ahead of rear* since *front* "passes" *rear* when the queue becomes empty. However, imagine you have enqueued n elements and no dequeues have occurred. All the array slots (0..n-1) are full so certainly the queue is full. On the other hand, front has not moved from its initial value of 0, so *front* is one slot ahead of *rear*, and the queue is empty. Obviously, we need a way to distinguish the two situations so we agree to give up one slot and make *front* the index of the location before the front of the queue. The *rear* index still points to the last element enqueued and (front + 1) modulo n is the next element to dequeue, if the queue is not empty. Consider a queue with elements elements [0..3]. This queue has four elements. Suppose front = 2, then this means that elements[3] contains the item at the front of the queue.

Initially the constructor would set **front = 0 and rear = 0**. Using the circular array implementation, the queue is **empty** if **front = rear**. The queue is **full** if **(rear + 1) % n** is equal to **front**. For example,



We cannot insert another element since (rear + 1) % 4 = 0 (=front). That is, the queue is full. But after a dequeue we could insert another element.

Lab Activity 01. Write the modified versions of enqueue and dequeue algorithms for circular queue and attach screenshots of both program code and output here:

```
#(Circular array) enqueue
```

```
#(Circular array) dequeue
```

Exercise : Redraw the above circular queue after all of the following operations have been completed:

```
item = q.dequeue()
q.enqueue(96)
item = q.dequeue()
```

Deque:

A **deque** is a double-ended queue in which elements can be both inserted and deleted from either the left or the right end of the queue. An implementation of a deque in Python is available in the collections module.

Below is a brief description of the methods used to modify a deque:

Insertion

append(item): Add an item to the right end.

appendleft(item): Add an item to the left end.

insert(index, value): Add an element with the specified value at the given index.

extend(list): This function is used to insert multiple values at the right end. It takes a list of values as an argument.

extendleft(list): This function is similar to `extend()`, but it reverses the list of values passed as the argument and then appends that list to the left end of the deque.

Deletion

`pop()`: Remove an element from the right end.

`popleft()`: Remove an element from the left end.

`remove(value)`: Remove the first occurrence of the mentioned value.

Miscellaneous

`count(value)`: Return the total number of occurrences of the given value.

`index(e, start, end)`: Search the given element from *start* to *finish* and return the index of the first occurrence.

`rotate(n)`: Rotate the deque *n* number of times. A positive value rotates it to the right, while a negative value rotates it to the left.

`reverse()`: Reverse the order of the deque.

Deques are particularly useful for implementing algorithms that require efficient insertion and removal operations. Here's an example showcasing the deque:

```
# Import collections module:
from collections import deque

# Initialize deque:
dq = deque([4,5,6])
# Append to the right:
dq.append(7)
print("Append 7 to the right: ", list(dq))

# Append to the left:
dq.appendleft(3)
print("Append 3 to the left: ", list(dq))

# Append multiple values to right:
dq.extend([8, 9, 10])
print("Append 8, 9 and 10 to the right: ", list(dq))

# Append multiple values to left:
dq.extendleft([1, 2])
print("Append 2 and 1 to the left: ", list(dq))

# Insert -1 at index 5
dq.insert(5, -1)
print("Insert -1 at index 5: ", list(dq))

# Pop element from the right end:
dq.pop()
print("Remove element from the right: ", list(dq))

# Pop element from the left end:
dq.popleft()
print("Remove element from the left: ", list(dq))
```

```
# Remove -1:
dq.remove(-1)
print("Remove -1: ", list(dq))

# Count the number of times 5 occurs:
i = dq.count(5)
print("Count the number of times 5 occurs: ", i)

# Return index of '7' if found between index 4 and 6:

index = dq.index(7, 4, 6)
print("Search index of number 7 between index 4 and 6: ",
index)

# Rotate the deque three times to the right:
dq.rotate(3)
print("Rotate the deque 3 times to the right: ", list(dq))

# Reverse the whole deque:
dq.reverse()
print("Reverse the deque: ", list(dq))
```

Understand the above code and determine the output?
Add screenshot here

Lab Activity:
Make use of Python collections module to implement stack and queues in Python.

Use `deque` as a stack (LIFO)
Use `deque` as a queue (FIFO)

Reference:
https://realpython.com/queue-in-python/#deque-double-ended-queue

Lab Tasks

Task 01 **You have to implement a waiting room management system in an emergency ward of a hospital. Your program will assign an Id number to a patient in a first come first serve basis. The lower the id, the sooner the service will be provided to the patient.**

Your program will contain the following methods:

> `RegisterPatient():` This method assigns an Id (which is auto-generated) to a patient and register him/her to the system.
>
> `ServePatient():` This method calls a patient to provide hospital service to him/her.
>
> `CancelAll():` This method cancels all appointments of the patients so that the doctor can go to lunch.
>
> `CanDoctorGoHome():` This method returns true if no one is waiting, otherwise, returns false.
>
> `ShowAllPatient():` This method shows all ids of the waiting patients in SORTED order. [Sorted according to their names]

Solve the above problem using an *array based queue*.

Important Note: Practice your knowledge of OOP with Python when creating a solution.

Task 02. `Palindrome-Checker`

An interesting problem that can be easily solved using the **deque** data structure is the classic palindrome problem.

A **palindrome** is a string that reads the same forward and backward, for example, *radar*, *toot*, and *madam*. Construct an algorithm to input a string of characters and check whether it is a palindrome.

The solution to this problem will use a deque to store the characters of the string. We will process the string from left to right and add each character to the rear of the deque. At this point, the deque will be acting very much like an ordinary queue. However, you can now make use of the dual functionality of the deque. The front of the deque will hold the first character of the string and the rear of the deque will hold the last character (see Figure 2).

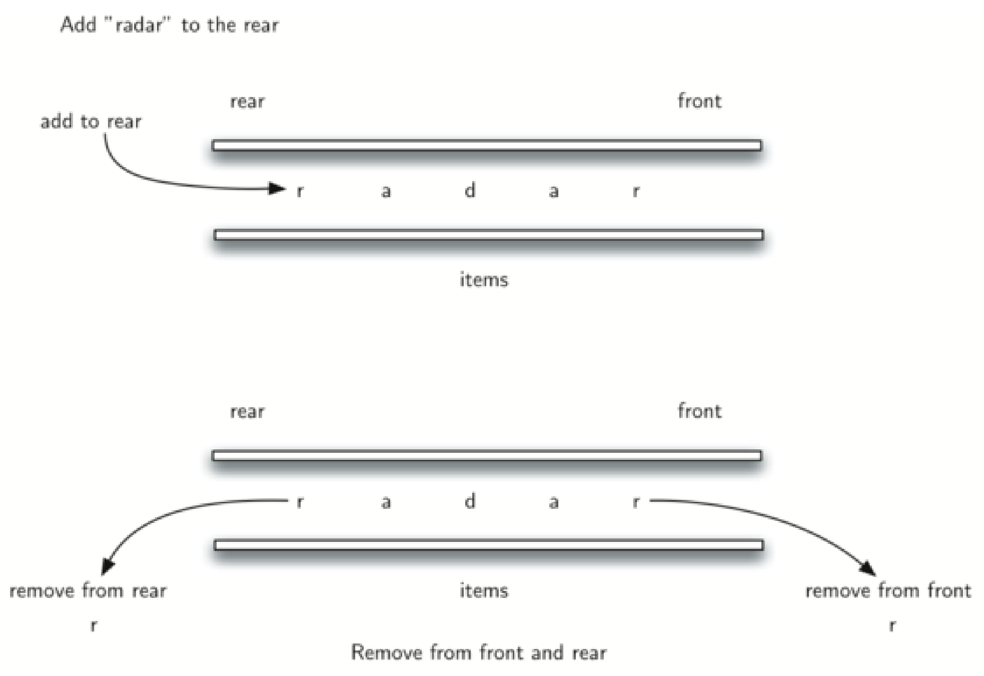


Figure 2: A Deque

Since we can remove both of them directly, we can compare them and continue only if they match. If we can keep matching first and the last items, we will eventually either run out of characters or be left with a deque of size 1 depending on whether the length of the original string was even or odd. In either case, the string must be a palindrome. Write a complete function for palindrome-checking and test you code for the following strings:

"Madam, I'm Adam"

"A man, a plan, a canal: Panama"

Task 3. Write a simple expression parser using stacks with the implementation of brackets with priority based arithmetic operators and integer such that the result of division should also be kept an integer.

1. Save the answers of the expression-1, expression-2 and so on in a 5 – element queue with the key-name Ans 1., Ans 2., and so on.
2. The console layout should be in the way shown below.

```
      Expression                    Results

          1.                        Ans 1.

          2.                        Ans 2.

          3.                        Ans 3.

          4.                        Ans 4.

          5.                        Ans 5.
```

3. In the start of program, the user will enter an expression as following:
   10 + 3 * 5 / (16 – 4)
4. As the user press enter, the expression will go to the place 1. In expression column and its result in Ans 1. place in results column.

Expression	Results
1. 10 + 3 * 5 / (16 – 4)	Ans 1. 15
2.	Ans 2.
3.	Ans 3.
4.	Ans 4.
5.	Ans 5.

5. In this way the user will enter up to 5 equations which will be displayed in expression column and their result in the results column
6. As soon as the user enters the 6th equation, the 1st equation and its answer should be removed, all the other equations should move a step up and the recently entered equation and its answer should occupy space number 5 in both columns and the same process should continue for further equations. Implement it using **queues.**
7. The console screen should have a constant display, i.e. it should not print the layout below the previous again and again. For this purpose use time.sleep() method

Precedence

```
–   ()                                        (High)
–  exp ^     R -> L
–  * or /     L -> R
–  + or -      L -> R                        (Low)
```

➔ Use only one type of bracket in the expression.
➔ Each expression must have all operators.

Grade Criteria

This lab is graded. Min marks: 0. Max marks: 30

Tasks Shown during Lab (At least class activity and Task 1 must be shown to LE during lab working) 10 marks	Modern Tool Usage 5 marks	Lab Ethics 5 Marks	Lab Report and Tasks Final Submission 5 marks	Lab Viva/Quiz 5 Marks
Total Marks: 30	Obtained Marks: _____			

Happy Coding!

Deliverables

Comment your program heavily. Intelligent comments and a clean, readable formatting of your code account for 20% of your grade.

You should submit your codes and report as a compressed zip file. It should contain all files used in the exercises for this lab.

The submitted file should be named cs250_firstname_lastname_lab08.zip