CS211 - Data Structures and Algorithms

Lab 9

Instructor: Dr. Sharaf Hussain
E-mail: shhussain@uit.edu

Semester: Fall, 2021

1 Objective

The purpose of this lab session is to implement **Queue** data structure and its applications.

2 Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

3 How to Submit

- Submit lab work in a single .py file on Google Classroom. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW04.py)
- Submit home work in a single .py file on Google Classroom. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_HW04.py)

Task 1 - Queue using python list

Implement the Queue data structure using python list.

```
#Queue ADT using Python list
class Queue:
    def __init__(self):
        self._qList = list()

    def isEmpty(self):
        return len(self) == 0

    def __len__(self):
        return len(self._qList)

    def enqueue(self, item):
        self._qList.append(item)

    def dequeue(self):
        assert not self.isEmpty(), "Queue is empty"
        self._qList.pop(0)
    def printQueue(self):
        print(self._qList)
```

Task 2 - Queue using circular array

Implement Queue data structure using circular array.

```
# Implementation of Queue Using circular array.
import sys
sys.path.append("../")
from Chapter2.myarray import Array

class Queue:
    def __init__(self, maxSize):
        self._count = 0 #count the number of item
        self._front = 0 #front set to 0 in the beginning
        self._back = maxSize - 1 #back at the end in the beginning
        self._qArray = Array(maxSize) #array to hold item

    #returns if the queue is empty
    def isEmpty(self):
        return self._count == 0

    #returns true if the queue is full
    def isFull(self):
        return self._count == len(self._qArray)

    #returns the no of itmes in the queue
    def __len__(self):
        return self._count

    #adds the given item to the queue
    def enqueue(self, item):
        assert not self.isFull(), "Queue filled up"
        maxSize = len(self._qArray)
        self._back = (self._back + 1) % maxSize
        self._qArray[self._back] = item
        self._count += 1

    #removes and returns the first item
    def dequeue(self):
        assert not self.isEmpty(), "Queue is empty"
        item = self._qArray[self._front]
        maxSize = len(self._qArray)
        self._front = (self._front + 1) % maxSize
        self._count -= 1
        return item
```

Task 3 - Queue using linked list

Implement Queue data structure using linked list.

```
#Implementation of the Queue ADT using a linked list
class Queue:
    def __init__(self):
        self._qhead = None
        self._qtail = None
        self._count = 0
    #Returns true if the queue is empty
    def isEmpty(self):
        return self._qhead is None
    #Returns the number of items
    def __len__(self):
        return self._count
    #adds the given item to the queue
    def enqueue(self, item):
        node = _QueueNode(item)
        if self.isEmpty():
            self._qhead = node
        else:
            self._qtail.next = node

        self._qtail = node
        self._count += 1
```

```
    #Removes and returns the first item
    def dequeue(self):
        assert not self.isEmpty(), "Can not be empty"
        node = self._qhead
        if self._qhead is self._qtail:
            self._qtail = None
        self._qhead = self._qhead.next
        self._count -= 1
        return node.item

class _QueueNode(object):
    def __init__(self, item):
        self.item = item
        self.next = None
```

Task 4 - Priority Queue - Python list

Implement Priority Queue data structure using python list.

```
#Unbounded Priority Queue ADT using list
class PriorityQueue:
    def __init__(self):
        """initlaizes a list for unbounded queue"""
        self._qList = list()

    def isEmpty(self):
        """True if queue is empty"""
        return len(self) == 0

    def __len__(self):
        return len(self._qList)
    #O(n) - but O(1) amortized cost
    def enqueue(self, item, priority):
        """New instance of storage class is created, and appended to the
 list"""
        entry = self._PriorityQEntry(item, priority)
        self._qList.append(entry)
    #O(n) as it transverses the list in worst possible case
    def dequeue(self):
        """Looks for the entry with highest priority and dequeue"""
        assert not self.isEmpty(), "Empty queue"
        index = 0
        highest = self._qList[index].priority
        for i in range(len(self)):
            if self._qList[i].priority < highest:
                highest = self._qList[i].priority
                index = i
        entry = self._qList.pop(index)
        return entry.item
    #__slots__ is good when many instances of the class is to be crated
    #the class should inherit object, and all of the inheritance
 shoulddelcare
    #__slot__ and not __dict__
    #arbirtray attributes can not be added in __slots__ unlike in __dict__
  though
    class _PriorityQEntry(object):
        __slots__ = 'item', 'priority'
        def __init__(self, item, priority):
            self.item = item
            self.priority = priority
```

Task 5 - Priority Queue - Linked List

Implement Priority Queue data structure using linked list.

Task 6 - Bounded Priority Queue - Linked List

Implement Bounded Priority Queue data structure using linked list.

Task 7 - Simulation - Printer Queue

Printers can be connected to the network and made available to many users. To manage multiple print jobs and to provide fair access, networked printers use print queues. Design, implement, and test a computer program to simulate a print queue that evaluates the average wait time.