Maira Usman

21b-011-se

In [5]:

```
class Queue:
    def __init__(self):
        self.items = list()

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        return self.items.pop()

    def print(self):
        print(self.items)
obj=Queue()
obj.enqueue(2)
obj.enqueue(6)
obj.enqueue(15)
obj.enqueue(27)
obj.print()
obj.dequeue()
obj.print()
```

```
[27, 15, 6, 2]
[27, 15, 6]
```

In [12]:

```
import ctypes
class Array:
     def __init__(self, size):
         assert size > 0, "Array size must be > 0"
         self._size = size
         #create the array structure using the ctypes module
         PyArrayType = ctypes.py_object * size
         self._elements = PyArrayType()
         #initialize each element using clear method of Array class
         self.clear(None)
     def __len__(self):
         return self._size
     def __getitem__(self, index):
         assert index >=0 and index < len(self), "Array subscript out of range!"
         return self._elements[index]
     def __setitem__(self, index, value):
         assert index >=0 and index < len(self), "Array subscript out of range!"
         self._elements[index] = value
     def clear(self, value):
         for i in range(len(self)):
             self._elements[i] = value
     def __iter__(self):
         return _ArrayIterator(self._elements)
class _ArrayIterator:
     def __init__(self, theArray):
         self._arrayRef = theArray
         self._curNdx = 0
     def __iter__(self):
         return self
     def __next__(self):
         if self._curNdx < len(self._arrayRef):
             entry = self._arrayRef[self._curNdx]
             self._curNdx += 1
             return entry
         else:
             raise StopIteration
class CircularQueue():
    # constructor
    def __init__(self, size): # initializing the class
        self.size = size
        # initializing queue with none
        self.queue = [None for i in range(size)]
        self.front = self.rear = -1
    def enqueue(self, data):
        # condition if queue is full
        if ((self.rear + 1) % self.size == self.front):
            print(" Queue is Full\n")
        # condition for empty queue
        elif (self.front == -1):
            self.front = 0
            self.rear = 0
            self.queue[self.rear] = data
        else:
            # next position of rear
            self.rear = (self.rear + 1) % self.size
            self.queue[self.rear] = data
    def dequeue(self):
        if (self.front == -1): # condition for empty queue
            print ("Queue is Empty\n")
```

```
        # condition for only one element
        elif (self.front == self.rear):
            temp=self.queue[self.front]
            self.front = -1
            self.rear = -1
            return temp
        else:
            temp = self.queue[self.front]
            self.front = (self.front + 1) % self.size
            return temp
    def display(self):
        # condition for empty queue
        if(self.front == -1):
            print ("Queue is Empty")
        elif (self.rear >= self.front):
            print("Elements in the circular queue are:",end = " ")
            for i in range(self.front, self.rear + 1):
                print(self.queue[i], end = " ")
            print ()
        else:
            print ("Elements in Circular Queue are:",end = " ")
            for i in range(self.front, self.size):
                print(self.queue[i], end = " ")
            for i in range(0, self.rear + 1):
                print(self.queue[i], end = " ")
            print ()
        if ((self.rear + 1) % self.size == self.front):
            print("Queue is Full")
# Driver Code
ob = CircularQueue(5)
ob.enqueue(14)
ob.enqueue(22)
ob.enqueue(13)
ob.enqueue(-6)
ob.display()
print ("Deleted value = ", ob.dequeue())
print ("Deleted value = ", ob.dequeue())
ob.display()
ob.enqueue(9)
ob.enqueue(20)
ob.enqueue(5)
ob.display()
```

```
Elements in the circular queue are: 14 22 13 -6
Deleted value =  14
Deleted value =  22
Elements in the circular queue are: 13 -6
Elements in Circular Queue are: 13 -6 9 20 5
Queue is Full
```

In [15]:

```
#Task 3  Queue using 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

    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
    def print(self):
        curnode=self._qhead
        while curnode:
            print(curnode.item,end=' ')
            curnode=curnode.next
class _QueueNode(object):
    def __init__(self, item):

        self.item = item
        self.next = None
q = Queue()
q.enqueue(2)
q.enqueue(7)
q.enqueue(27)
q.enqueue(15)
q.print()
print()
q.dequeue()
q.print()
#Task 4 Priority Queue - Python list
```

```
2 7 27 15
7 27 15
```

In [20]:

```
#Task 4 Priority Queue - Python list
class PriorityQueue :
    def __init__ ( self ) :
        self . _qList = list ()
    def isEmpty ( self ) :
        return len( self ) == 0
    def __len__ ( self ) :
        return len( self . _qList )
    def enqueue ( self , item , priority ) :
        entry = self . _PriorityQEntry ( item , priority )
        self . _qList . append ( entry )
    def dequeue ( self ) :
        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 . i
    class _PriorityQEntry ( object ) :
        def __init__ ( self , item , priority ) :
            self . item = item
            self . priority = priority
    def print(self):
        for entry in self._qList:
            print(f"Item: {entry.item}, Priority: {entry.priority}")

obj=PriorityQueue()
obj.enqueue("apple",1)
obj.enqueue("carrot",3)
obj.enqueue("honey",2)
obj.print()
```

```
Item: apple, Priority: 1
Item: carrot, Priority: 3
Item: honey, Priority: 2
```

In [21]:

```
class Node:
    def __init__(self, data, priority):
        self.data = data
        self.priority = priority
        self.next = None

class PriorityQueue:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head is None
    def enqueue(self, data, priority):
        new_node = Node(data, priority)
        if self.is_empty() or priority < self.head.priority:
            new_node.next = self.head
            self.head = new_node
        else:
            current = self.head
            while (current.next is not None) and \
                  (current.next.priority <= priority):
                current = current.next
            new_node.next = current.next
            current.next = new_node
    def dequeue(self):
        if self.is_empty():
            return None
        data = self.head.data
        self.head = self.head.next
        return data
    def print(self):
        curnode=self.head
        while curnode:
            print(f"Item: {curnode.data}, Priority: {curnode.priority}")
            curnode=curnode.next
queue = PriorityQueue()
queue.enqueue('A', 2)
queue.enqueue('B', 1)
queue.enqueue('C', 3)
queue.print()
print("After Deletion")
queue.dequeue()
queue.print()
```

```
Item: B, Priority: 1
Item: A, Priority: 2
Item: C, Priority: 3
After Deletion
Item: A, Priority: 2
Item: C, Priority: 3
```

In [25]:

```
#Task 6 Bounded Priority Queue - Linked List
class PriorityQueueNode:
    def __init__(self, value, pr):
        self.data = value
        self.priority = pr
        self.next = None
class PriorityQueue:
    def __init__(self):
        self.front = None
    def isEmpty(self):
        return True if self.front == None else False
    def enqueue(self, value, priority):
        assert priority<=10 ,"Priority must not be greater than 10"
        if self.isEmpty() == True:
            self.front = PriorityQueueNode(value,priority)
            return 1
        else:
            if self.front.priority > priority:
                newNode = PriorityQueueNode(value,priority)
                newNode.next = self.front
                self.front = newNode
                return 1
            else:
                temp = self.front
                while temp.next:
                    if priority <= temp.next.priority:
                        break
                    temp = temp.next
                newNode = PriorityQueueNode(value,priority)
                newNode.next = temp.next
                temp.next = newNode
                return 1
    def dequeue(self):
        if self.isEmpty() == True:
            return
        else:
            self.front = self.front.next
            return 1

    def peek(self):
        if self.isEmpty() == True:
            return
        else:
            return self.front.data

    def print(self):

        if self.isEmpty() == True:
            return "Queue is Empty!"
        else:
            temp = self.front
            while temp:
                print(temp.data)
                temp = temp.next

obj1 = PriorityQueue()
obj1.enqueue(3, 1)
obj1.enqueue(2, 2)
obj1.enqueue(1, 3)
```

```
obj1.enqueue(6, 5)
print("bounded queue after traverse")
obj1.print()
obj1.dequeue()
obj1.enqueue(5,4)
print("bounded queue after traverse")
obj1.print()
```

```
bounded queue after traverse
3
2
1
6
bounded queue after traverse
2
1
5
6
```

In [28]:

```
#Task 7 Simulation - Printer Queue
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        return self.items.pop()

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

import random
class Printer:
    def __init__(self, ppm):
        self.pagerate = ppm
        self.currentTask = None
        self.timeRemaining = 0

    def tick(self):
        if self.currentTask != None:
            self.timeRemaining = self.timeRemaining - 1
            if self.timeRemaining <= 0:
                self.currentTask = None

    def busy(self):
        if self.currentTask != None:
            return True
        else:
            return False

    def startNext(self, newtask):
        self.currentTask = newtask
        self.timeRamaining = newtask.getPages() * 60/self.pagerate

class Task:
    def __init__(self, time):
        self.timestamp = time
        self.pages = random.randrange(1, 21)

    def getStamp(self):
        return self.timestamp

    def getPages(self):
        return self.pages

    def waitTime(self, currenttime):
        return currenttime - self.timestamp

def simulation(numSecs, pagesPerMins):
    labprinter = Printer(pagesPerMins)
    printQueue = Queue()
    waitingtimes = []
```

```
    for currentSec in range(numSecs):
        if newPrintTask():
            task = Task(currentSec)
            printQueue.enqueue(task)

    if (not labprinter.busy()) and (not printQueue.isEmpty()):
        nexttask = printQueue.dequeue()
        waitingtimes.append(nexttask.waitTime(currentSec))
        labprinter.startNext(nexttask)

    labprinter.tick()

    averageWait = sum(waitingtimes)/len(waitingtimes)
    print('Average Wait Time: %6.2f secs ,  tasks ramaining.: %3d'%(averageWait, printQueue

def newPrintTask():
    num = random.randrange(1, 181)
    if num == 180:
        return True
    else:
        return False

for i in range(10):
    simulation(4600, 5)
```

```
Average Wait Time: 4302.00 secs ,  tasks ramaining.:  33
Average Wait Time: 3631.00 secs ,  tasks ramaining.:  19
Average Wait Time: 4250.00 secs ,  tasks ramaining.:  23
Average Wait Time: 3042.00 secs ,  tasks ramaining.:  20
Average Wait Time: 4257.00 secs ,  tasks ramaining.:  25
Average Wait Time: 3935.00 secs ,  tasks ramaining.:  22
Average Wait Time: 4579.00 secs ,  tasks ramaining.:  24
Average Wait Time: 4421.00 secs ,  tasks ramaining.:  29
Average Wait Time: 4421.00 secs ,  tasks ramaining.:  28
Average Wait Time: 4416.00 secs ,  tasks ramaining.:  19
```

In [ ]:

```
```