Al-Mustaqbal University
College of Sciences
Intelligent Medical System Department

جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Subject: Data Structure

Class: Second

Lecturer: Prof.Dr. Mehdi Ebady Manaa

Lecture: (6)

Priority Queue

PRIORITY QUEUE CONCEPT

A priority queue is best understood in comparison with a stack and a queue. To see this, imagine a supermarket checkout, where customers, each with a certain number of items in the shopping cart, arrive at the checkout counter:

ItemsInCart	Customer	
6	Mary	//last to arrive
12	Joe	
4	Jill	
9	Pete	
15	Stacy	
7	Bev	//first to arrive

CHECKOUT COUNTER

Suppose also that the entries <6, Mary>, <12, Joe> … <7, Bev> are placed in a data container, to reflect order of arrival, with <7, Bev> first in, <15, Stacy> next in, and so on, and <6, Mary> in last.

If the cashier serves the customers in order of arrival, that is, <7, Bev> first and <6, Mary>last, we have a conventional queue, i.e. First In, First Out, or **FIFO**.

If the cashier serves the customers starting with entries <6, Mary>, and ending with <7, Bev>, we have a stack, i.e. Last In, First Out, or **LIFO**.

If the cashier serves the customers with entries in the **order** <4, Jill>, <6, Mary>, <7, Bev>, … ending with <15, Stacy>, that is, service in order of lowest number of shopping items, we have a **priority queue**, i.e. The entry inserted with the **lowest priority key**, no matter when inserted, is the first entry out.
In this example, the first data item of each entry, the number of shopping items, serves as the **priority key**.
Notice the technical term entry. A priority queue consists of a **set of entries** into the queue, each entry consisting of a **priority key and a value**.

Applications

- Scheduling jobs on a workstation holds jobs to be performed and their priorities. When a job is finished or interrupted, highest-priority job is chosen using Extract-Max. New jobs can be added using Insert function.
- Operating System Design – resource allocation
- Data Compression -Huffman algorithm
- Discrete Event simulation

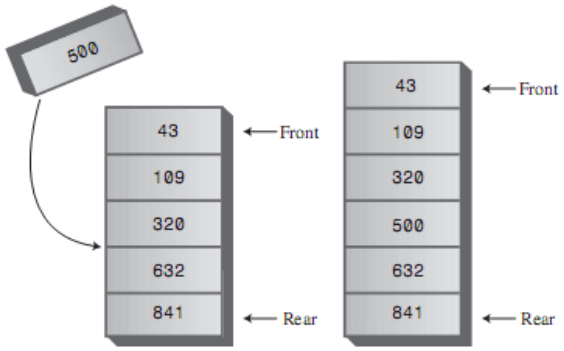
(1) Insertion of time-tagged events (time represents a priority of an event -- low time means high priority)

(2) Removal of the event with the smallest time tag

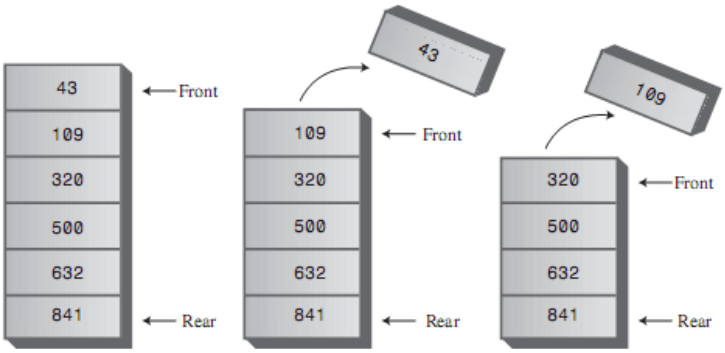
Implementation

- Linked Lists
- Using a binary Heap – a special binary tree with heap property

We show **Front** and **Rear** arrows to provide a comparison with an ordinary queue, but they're not really necessary. The algorithms know that the **front** of the queue is always at the **top** of the array at **nItems-1**, and they insert items in order, **not at the rear**. Figure below shows the operation of the PriorityQ class methods.



New item inserted in priority queue



Two items removed from front of priority queue

Key Comparison Method

Normally the entry with the **highest priority** has the **lowest priority key value**, and is extracted from the priority queue first.

That means that we need a way to **compare the key values**, so that we can say if the key of one entry is greater or less than the key of another entry, and which key has the lowest value and which the highest value.

The key comparison method may be very simple, based on integer values, as in the case of number of shopping items above.

The Priority Queue ADT

A priority queue ADT will be implemented as a container of some kind that can support the methods below.

constructor
Create a new, empty queue.

`insert`
Add a new item to the queue.

`remove`
Remove and return an item from the queue. The item that is returned is the one with the highest priority.

`empty`
Check whether the queue is empty.

Sorting With a Priority Queue

We can look at a priority queue as a black box. You can put entries into it, using insert(), in any key order, take out a few entries, using removeMin(), put in some more and so on, as if using a stack. But no matter how many we put in and take out, removeMin() always delivers the entry in the queue with the lowest key value. **This is obviously useful for sorting.**

Suppose we want to **sort a set** of entries (each made up of a key and a value) in ascending order:

Step 1. Use insert() to insert, in any order, all the entries into the priority queue.
Step 2. Use removeMin() to extract the entries from the queue, and print them, or place them in an array. The entries are now sorted.

Class PriorityQueue

The **insert()** method checks whether there are any items; if not, it inserts one at index **0**. **Otherwise**, it starts at the **top** of the array and shifts existing items upward until it finds the place where the **new item** should go. Then it inserts the item and **increments nItems**. Note that if there's any chance the priority queue is full, you should check for this possibility with **isFull()** before using insert().
The front and rear fields aren't necessary as they were in the Queue class because, as we noted, **front** is always at **nItems-1** and **rear** is always at **0**.
The **remove()** method is simplicity itself: It **decrements nItems** and returns the item from the **top**

of the array. The **isEmpty()** and **isFull()** methods check if **nItems** is **0** or **maxSize**, respectively

```
class PriorityQueue:
   def __init__(self):
      self.queArray[] =
      self.nItems = 0
```

```
    def insert(self, item):
        if self.nItems == 0:
            self.queArray.append(item)
        else:
            j = self.nItems - 1
            while j >= 0:
                if item > self.queArray[j]:
                    self.queArray[j + 1] = self.queArray[j]
                else:
                    break
                j -= 1
            self.queArray[j + 1] = item
        self.nItems += 1

    def remove(self):
        if self.nItems > 0:
            self.nItems -= 1
            return self.queArray.pop(self.nItems)
        else:
            raise IndexError("Priority queue is empty")

 #Example usage of PriorityQueue
pq = PriorityQueue()
pq.insert(3)
pq.insert(1)
pq.insert(4)
pq.insert(2)

print("Removed items in priority order:")
while pq.nItems > 0:
    print(pq.remove())
```

Q1. What is the primary characteristic that distinguishes a priority queue from a regular queue or stack?
a) It uses a stack-like Last In, First Out (LIFO) approach.
b) It follows a First In, First Out (FIFO) approach.
c) Items are served based on a priority key, not their arrival order.
d) It can only store items of equal priority.
Answer: c) Items are served based on a priority key, not their arrival order.

Q2. In a priority queue, what serves as the priority key for each entry in the queue, as described in the example?
a) Customer names
b) Number of shopping items
c) Order of arrival
d) Checkout counter number
Answer: b) Number of shopping items

Q3. Which of the following applications is NOT a suitable use case for a priority queue?
a) Scheduling jobs based on their priority.
b) Sorting a list of elements in ascending order.
c) Managing resource allocation in an operating system.
d) Simulating events with equal priority.
Answer: b) Sorting a list of elements in ascending order.

Q4. What is the purpose of the `insert` operation in a priority queue?
a) To add a new item to the end of the queue.
b) To remove an item from the queue.
c) To serve the item with the highest priority.
d) To remove the last item that arrived.
Answer: a) To add a new item to the end of the queue.**

Q5. Which data structure can be used to efficiently implement a priority queue with operations like `insert` and `remove`?
a) Array
b) Linked List
c) Binary Heap
d) Queue
Answer: c) Binary Heap