Data Structures - Queues

Dr. TGI Fernando [1] [2]

[1] Email: tgi.fernando@gmail.com
[2] URL: http://tgifernando.wordpress.com/

Queues

A data structure in which first item inserted is the first to be removed (First-In-First-Out, FIFO).

Additions are made at the end (or tail) of the queue while removals are made from the front (or head) of the queue.

Queues (Contd.)

Use as programmer's tool as stacks.

Use in real-world situations also.

- People waiting in line at a bank.
- Airplanes waiting to take off.
- Data packets waiting to be transmitted over the Internet.

There are various queues quietly doing their job in your computers (or the networks) operating system.

- Printer queue - where print jobs wait for the printer to be available.
- Keystroke data as you type at the keyboard.

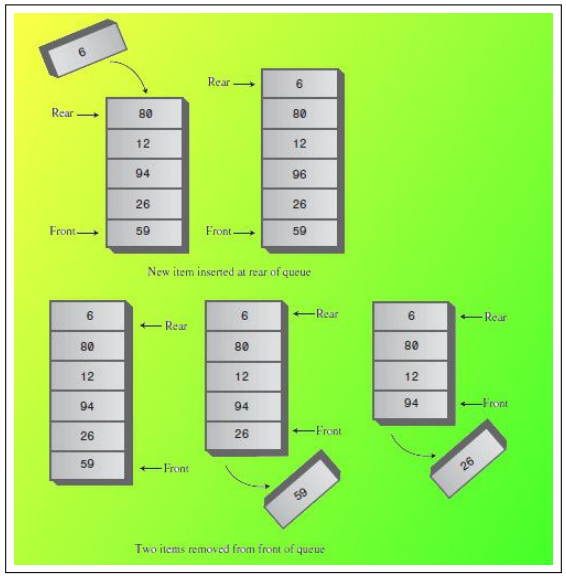
Primary queue operations

The two basic queue operations are inserting an item, which is placed at the rear of the queue, and removing an item, which is taken from the front of the queue.

Terms:

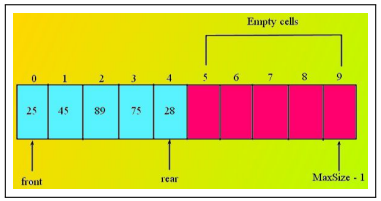
- Insert - put, add or enque
- Remove - delete, get or deque
- Front - head
- End - tail

Primary queue operations (Contd.)



Array implementation

Need two integer pointers to keep the `rear` and the `front`.

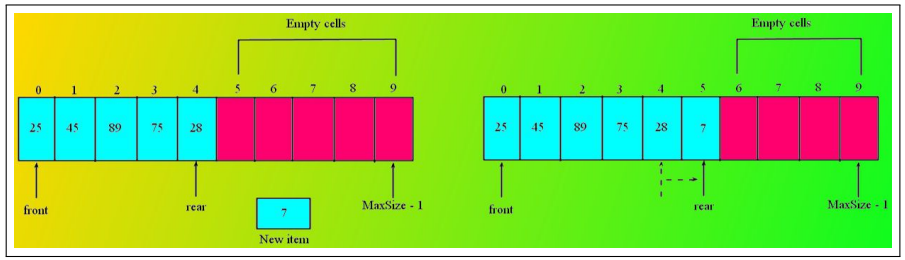


Initially `rear = -1` and `front = 0`.
To **insert** an item, `rear` of the queue is incremented by 1 and put the new item at `rear`.
To **delete** the item at the front, increment `front` by 1 (output - deleted item).
Peek - returns the value of the item at the front of the queue without removing the item.

Array implementation (Contd.)

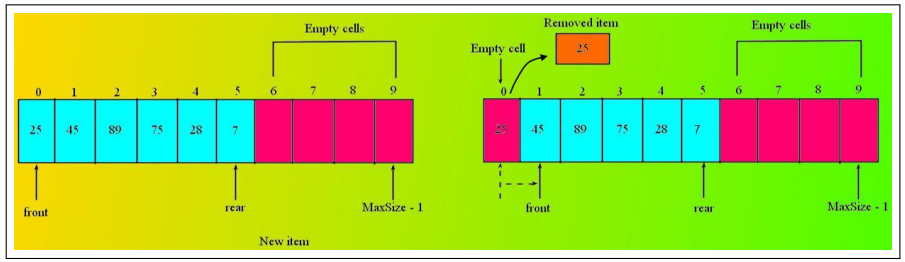
When removing item, it is not efficient to move items; instead we keep all the items in the same place and move the `front` and `rear` of the queue.

When you insert an item in the queue, `rear` moves toward higher numbers in the array.



Array implementation (Contd.)

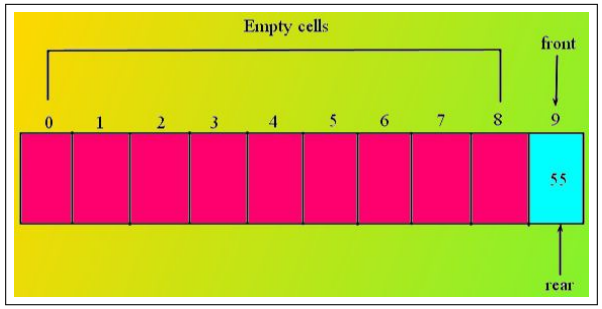
When you remove an item, `front` also moves toward `maxSize-1`.



Note: Removed item is still in the array (memory), but would not be accessible because `front` had moved past it.

Trouble with this implementation

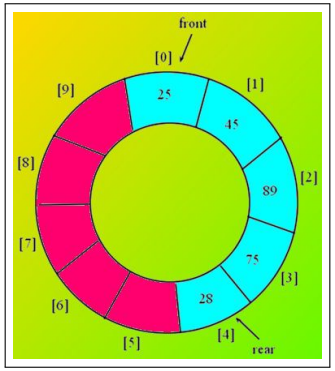
Trouble - `rear` of the queue is at the end of the array (the highest index) at some point. Even if there are empty cells at the beginning of the array, it is not possible to insert new items because `rear` can't go any further with this implementation.



Circular Queue

To avoid the problem of not being able to insert items into the queue when it is not full. `front` and `rear` wrap around to the beginning of the array.

The result is a **circular queue**.



Array implementation

Dealing with wraparound

```
rear = (rear + 1) % maxSize;
front = (front + 1) % maxSize;
```

Initially,
`rear = -1` and `front = 0`

```
class QueueArray {
  private int maxSize;
  private int[] queArray;
  private int front;
  private int rear;
  private int nItems;
  //--------------------------------------
  // ...
}
```

Operations

The `insert()` Method

- Assumes that the queue is not full.
- Before calling the `insert()` method, call `isFull()`.
- Use wraparound method to increment the value of `rear`.

The `remove()` Method

- Assumes that the queue is not empty.
- Call `isEmpty()` to ensure this is true before calling `remove()`.

The `peek()` Method - returns the value of `front`.

The `isEmpty()`, `isFull()`, and `size()` Methods
The `isEmpty()`, `isFull()`, and `size()` methods all rely on the `nItems` field, respectively checking if its 0, if its `maxSize`, or returning its value.

Lab Work 07 - `QueueArray` class

1. Complete the coding of the `QueueArray` class.
2. Write a class (QueueArrayApp) to use the methods defined in `QueueArray`.

More on queues

Efficiency of queues

As with a stack, items can be inserted and removed from a queue in O(1) time.

Dequeues

- Double-ended queue.
- Can insert items at either end [`insertLeft()` and `insertRight()`] and delete them from either end [`deleteLeft()` and `deleteRight()`].

Priority Queues

- More specialized data structure than stacks and queue.
- A priority queue has a front and a rear.
- Items are removed from the front.
- However, insertion of items are different from an ordinary queue;
  - Items are ordered by key value so that the item with the lowest key (or the highest key) is always at the front.
  - When items are inserted, they need to be inserted into the proper position (not at the rear).
- Programmer's tool

Example - Mail Sorting

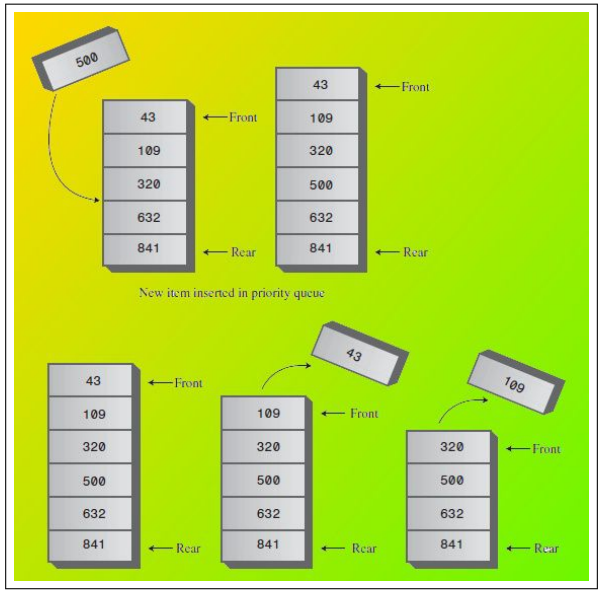
Every time the postman hands you a letter, you insert it into your pile of pending letters according to its priority.

- If it must be answered immediately (the phone company is about to disconnect your modem line), it goes on top.
- If it can wait for a leisurely answer (a letter from your mother-in-law), it goes on the bottom.
- Letters with intermediate priorities are placed in the middle; the higher the priority, the higher their position in the pile.
- The top of the pile of letters corresponds to the front of the priority queue.

Implementation

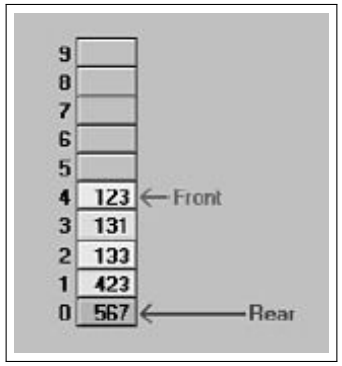
- Needs quick access to the most prioritised key.
- Also needs to provide fairly quick insertion.
- Implemented with a data structure called a 'heap'.
- Here we implement by a simple array.
- Suffers from slow insertion, but it's simpler.
- Appropriate when the number of items isn't high or insertion speed isn't critical.

Implementation (Contd.)



Implementation (Contd.)

- No wraparound - wouldn't improve the situation.
- Insertion is slow - need the proper in-order position.
- Deletion is fast.



`Front = nItems - 1` and `Rear = 0`

Implementation (Contd.)

Deletion

- The item to be removed is always at the top of the array, so removal is quick and easy.
- The item is removed and the `Front` arrow moves down to point to the new top of the array.
- No shifting or comparisons are necessary.

Insertion - Items are inserted in-order, not at the rear.

Peek - Peek the front item (without removing it).

PriorityQ class

```
class PriorityQ {
  private int maxSize;
  private int[] qArray;
  private int nItems;
  //--------------------------------------
  public PriorityQ(int s) { // constructor
    maxSize = s;
    qArray = new long[maxSize];
    nItems = 0;
  }
  public void insert(long item) {...}
  public int remove() {...}
  public long peekMin() {...}
  public boolean isEmpty() {...}
  public boolean isFull() {...}
}
```