UNIVERSIDAD NACIONAL DE COLOMBIA

Estructuras de Datos

Sesión 7
Queue Data Structure

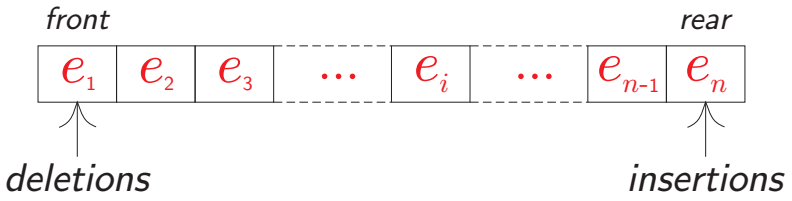
Yoan Pinzón

Table of Content Session 7

Queue Data Structure

A **queue** is a special case of linear list where insertions and deletions take place at *different* ends

rear: end at which a new element is added.
front: end at which an element is deleted.



In other words, a queue is a FIFO (first-in-first-out) list.

The Abstract Data Type

AbstractDataType Queue

{

instances: linear list of elements; one end is called the *front*; the other is the *rear*

operations:

`isEmpty()`: return `true` if the queue is empty, `false` otherwise

`getFrontElement()`: return the front element

`getRearElement()`: return the rear element

`put(`x`)`: add element x at the rear

`remove()`: remove an element from the front and return it

}

Interface Definition of Queue

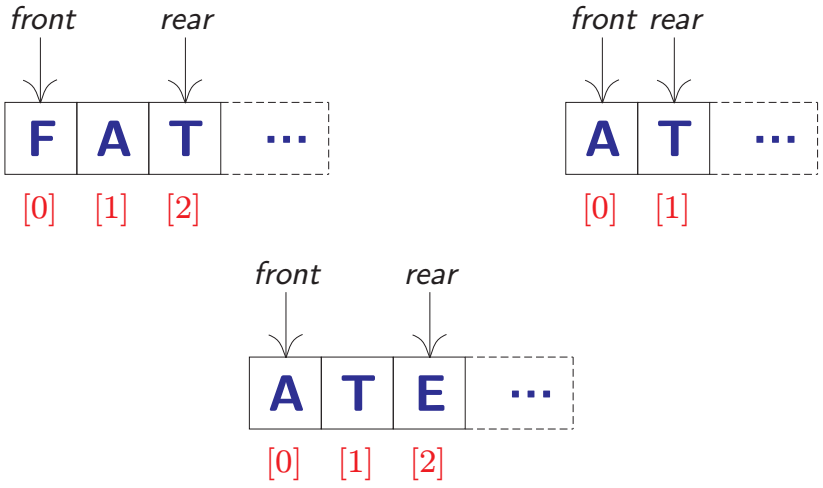
```
1  package unal.datastructures;
2
3  public interface Queue<T>
4  {
5     boolean isEmpty( );
6     T getFrontElement( );
7     T getRearElement( );
8     void put( T theObject );
9     T remove( );
10 }
```


Queue Data Structure

Array-based Representation

We can use three different approaches:

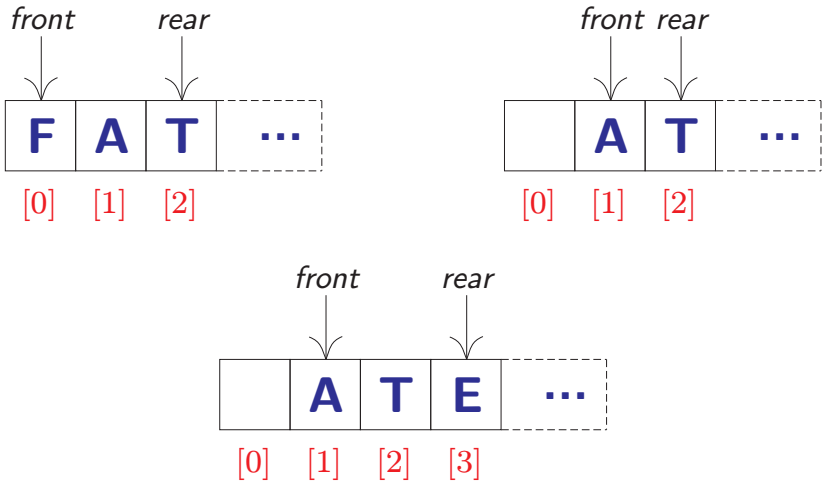
1) Using the formula $\boxed{location(i) = i}$



- **Empty queue:** rear = -1
- **Addition:**
  rear = rear + 1
  queue[rear] = new_element
  $\Theta(1)$ time
- **Deletion:**
  Shift all elements one position to the left.
  $\Theta(n)$ time

2) Using the formula

$$location(i) = location(\text{front element}) + i$$

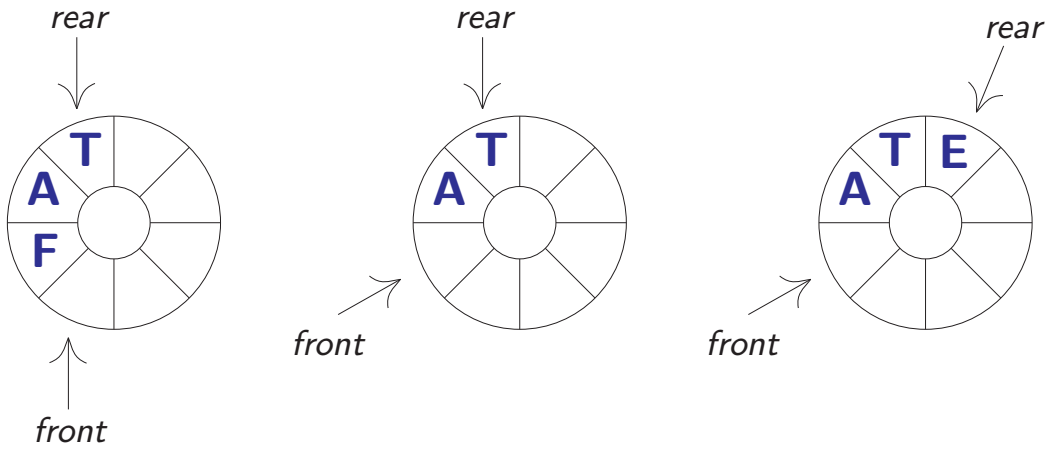


- **Empty queue:** rear < front
- front = $location$(front element)
- rear = $location$(last element)
- **Deletions & Insertions:** $\Theta(1)$ time

What happens if rear = queue.length and front > 0 ?

3) Using the formula:

$$location(i) = (location(\text{front element}) + i)\%queue.length$$



front points one position before the position of the first element in the queue.

- **Empty queue:** front = rear
(initially front=rear=0)
- **Full queue:** (rear+1)%queue.length = front

This is also called a **Circular Queue**

Class Definition of ArrayQueue

```
package unal.datastructures;

public class ArrayQueue<T> implements Queue<T>
{
   // fields
   int front; // one counterclockwise from first element
   int rear;  // position of rear element of queue
   T[] queue; // element array

   // constructors
   public ArrayQueue( int initialCapacity ) { /* ... */ }
   public ArrayQueue( ) { /* ... */ }

   // methods
   public boolean isEmpty( ) { /* ... */ }
   public T getFrontElement( ) { /* ... */ }
```

```
    public T getRearElement( ) { /* ... */ }
    public void put( T theElement ) { /* ... */ }
    public T remove( ) { /* ... */ }
    public static void main( String[] args ) { /* ... */ }
}
```


constructors

```
13  /** create a queue with the given initial capacity */
14  @SuppressWarnings( "unchecked" )
15  public ArrayQueue( int initialCapacity )
16  {
17     if( initialCapacity < 1 )
18        throw new IllegalArgumentException
19              ( "initialCapacity must be >= 1" );
20     queue = ( T[] ) new Object[ initialCapacity + 1 ];
21     front = rear = 0;
22  }

24  /** create a queue with initial capacity 10 */
25  public ArrayQueue( )
26  {
27     this( 10 );
28  }
```

isEmpty

```
31  /** @return true iff queue is empty */
32  public boolean isEmpty( )
33  {
34     return front == rear;
35  }
```


getFrontElement

```
37  /** @return front element of queue
38   * @return null if queue is empty */
39  public T getFrontElement( )
40  {
41     if( isEmpty( ) ) return null;
42     else return queue[ ( front + 1 ) % queue.length ];
43  }
```

getRearElement

```
45  /** @return rear element of queue
46   * @return null if the queue is empty */
47  public T getRearElement( )
48  {
49     if( isEmpty( ) ) return null;
50     else return queue[ rear ];
51  }
```


put

```
53  /** insert theElement at the rear of the queue */
54  @SuppressWarnings( "unchecked" )
55  public void put( T theElement )
56  {
57     if( ( rear + 1 ) % queue.length == front )
58     {  // double array size
59        // allocate a new array
60        T[] newQueue = ( T[] ) new Object [ 2 * queue.length ];

62        // copy elements into new array
63        int start = ( front + 1 ) % queue.length;
64        if( start < 2 )
65           // no wrap around
66           System.arraycopy( queue, start, newQueue, 0, ↙
               ↳ queue.length - 1 );
67        else
68        {  // queue wraps around
69           System.arraycopy( queue, start, newQueue, 0, ↙
               ↳ queue.length - start );
```

```
70            System.arraycopy( queue, 0, newQueue, queue.length - start, rear + 1 );
71         }
73         // switch to newQueue and set front and rear
74         front = newQueue.length - 1;
75         rear = queue.length - 2; // queue size is queue.length - 1
76         queue = newQueue;
77      }
79      // put theElement at the rear of the queue
80      rear = ( rear + 1 ) % queue.length;
81      queue[ rear ] = theElement;
82   }
```


remove

```
84   /** remove an element from the front of the queue
85    * @return removed element
86    * @return null if the queue is empty */
87   public T remove( )
88   {
89      if( isEmpty( ) ) return null;
90      front = ( front + 1 ) % queue.length;
91      T frontElement = queue[ front ];
92      queue[ front ] = null; // enable garbage collection
93      return frontElement;
94   }
```

main

```
96    /** test program */
97    public static void main( String[] args )
98    {
99       int x;
100      ArrayQueue<Integer> q = new ArrayQueue<>( 3 );

102      // add a few elements
103      q.put( new Integer( 1 ) );
104      q.put( new Integer( 2 ) );
105      q.put( new Integer( 3 ) );
106      q.put( new Integer( 4 ) );

108      // remove and add to test wraparound array doubling
109      q.remove( );
110      q.remove( );
111      q.put( new Integer( 5 ) );
112      q.put( new Integer( 6 ) );
113      q.put( new Integer( 7 ) );
114      q.put( new Integer( 8 ) );
```



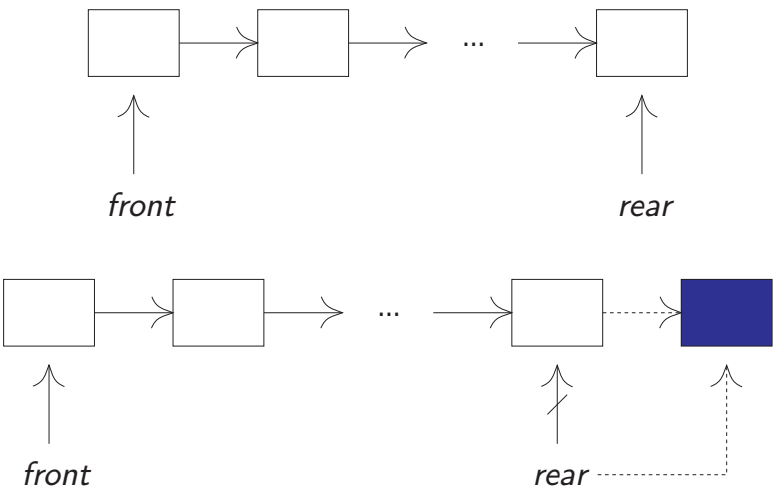
```
115      q.put( new Integer( 9 ) );
116      q.put( new Integer( 10 ) );
117      q.put( new Integer( 11 ) );
118      q.put( new Integer( 12 ) );

120      // delete all elements
121      while ( !q.isEmpty( ) )
122      {
123         System.out.println( "Rear element is " + q.getRearElement( ↙
           ↳ ) );
124         System.out.println( "Front element is " + ↙
           ↳ q.getFrontElement( ) );
125         System.out.println( "Removed the element " + q.remove( ) );
126      }
127   }
```

Compiling `ArrayQueue.java`

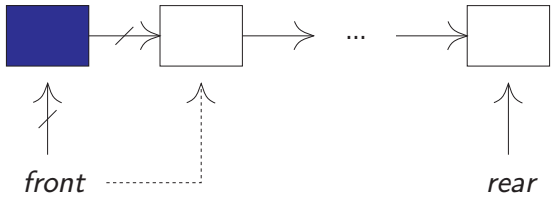
```
C:\2016699\code> javac unal\datastructures\ArrayQueue.java ↵
C:\2016699\code> java unal.datastructures.ArrayQueue ↵
Rear element is 12
Front element is 3
Removed the element 3
Rear element is 12
Front element is 4
Removed the element 4
Rear element is 12
Front element is 5
Removed the element 5
Rear element is 12
Front element is 6
Removed the element 6
Rear element is 12
Front element is 7
Removed the element 7
Rear element is 12
Front element is 8
Removed the element 8
Rear element is 12
Front element is 9
Removed the element 9
Rear element is 12
Front element is 10
Removed the element 10
Rear element is 12
Front element is 11
Removed the element 11
Rear element is 12
Front element is 12
Removed the element 12
```


Queue Data Structure

Linked Representation



(a) Addition



(b) Deletion

- **Empty queue:** front = null
- **Deletions & Insertions:** $\Theta(1)$ time

Class Definition of LinkedQueue

```
package unal.datastructures;

public class LinkedQueue<T> implements Queue<T>
{
   // fields
   protected ChainNode<T> front;
   protected ChainNode<T> rear;

   // constructor
   public LinkedQueue( ) { /* ... */ }

   // methods
   public boolean isEmpty( ) { /* ... */ }
   public T getFrontElement( ) { /* ... */ }
   public T getRearElement( ) { /* ... */ }
   public void put( T theElement ) { /* ... */ }
```

```
    public T remove( ) { /* ... */ }
    public static void main( String[] args ) { /* ... */ }
}
```


constructor

```
12  /** create an empty queue */
13  public LinkedQueue( )
14  {
15     front = rear = null;
16  }
```

isEmpty

```
19    /** @return true iff queue is empty */
20    public boolean isEmpty( )
21    {
22       return front == null;
23    }
```


getFrontElement

```
25    /** @return the element at the front of the queue
26     * @return null if the queue is empty */
27    public T getFrontElement( )
28    {
29       return isEmpty( ) ? null : front.element;
30    }
```

getRearElement

```
32  /** @return the element at the rear of the queue
33   * @return null if the queue is empty */
34  public T getRearElement( )
35  {
36     return isEmpty( ) ? null : rear.element;
37  }
```


put

```
39  /** insert theElement at the rear of the queue */
40  public void put( T theElement )
41  {
42     ChainNode<T> p = new ChainNode<T>( theElement, null );

44     if( front == null ) front = p; // empty queue
45     else rear.next = p; // nonempty queue

47     rear = p;
48  }
```

remove

```
50  /** remove an element from the front of the queue
51   * @return removed element
52   * @return null if the queue is empty */
53  public T remove( )
54  {
55     if( isEmpty( ) ) return null;
56     T frontElement = front.element;
57     front = front.next;
58     if( isEmpty( ) ) rear = null; // enable garbage collection

60     return frontElement;
61  }
```


main

```
63  /** test program */
64  public static void main( String[] args )
65  {
66     int x;
67     LinkedQueue<Integer> q = new LinkedQueue<>( );

69     // add a few elements
70     q.put( new Integer( 1 ) );
71     q.put( new Integer( 2 ) );
72     q.put( new Integer( 3 ) );
73     q.put( new Integer( 4 ) );

75     // delete all elements
76     while ( !q.isEmpty( ) )
77     {
78        System.out.println( "Rear␣element␣is␣" + q.getRearElement( ↙
```

```
            ↳ ) );
79          System.out.println( "Front␣element␣is␣" + ↙
              ↳ q.getFrontElement( ) );
80          System.out.println( "Removed␣the␣element␣" + q.remove( ) );
81      }
82  }
```


Compiling `LinkedQueue.java`

```
C:\2016699\code> javac unal\datastructures\LinkedQueue.java ↵
C:\2016699\code> java unal.datastructures.LinkedQueue ↵
Rear element is 4
Front element is 1
Removed the element 1
Rear element is 4
Front element is 2
Removed the element 2
Rear element is 4
Front element is 3
Removed the element 3
Rear element is 4
Front element is 4
Removed the element 4
```