Lecture 7: Queues

Bohyung Han
CSE, POSTECH
bhhan@postech.ac.kr

Queues

What is queue?
- The Queue ADT stores arbitrary objects.
- An ordered list in which insertions and deletions are made at different sides, called rear and front, respectively.

Properties
- First-In-First-Out (FIFO) scheme
- Typically implemented with array or linked list

Queue class in Java
- http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html

Queue Operations

Main operations:
- enqueue(object): inserts an element at the “rear” of the queue
- object dequeue(): removes and returns the element at the “front” of the queue

Others:
- object peek(): returns the element at the “front” without removing it
- int size(): returns the number of elements stored
- boolean isEmpty(): indicates whether no elements are stored

An Example

<table>
<thead>
<tr>
<th>Operation</th>
<th>Output</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>enqueue(5)</td>
<td>–</td>
<td>[5]</td>
</tr>
<tr>
<td>enqueue(3)</td>
<td>–</td>
<td>[5, 3]</td>
</tr>
<tr>
<td>dequeue()</td>
<td>5</td>
<td>[3]</td>
</tr>
<tr>
<td>enqueue(7)</td>
<td>–</td>
<td>[3, 7]</td>
</tr>
<tr>
<td>dequeue()</td>
<td>3</td>
<td>[7]</td>
</tr>
<tr>
<td>peek()</td>
<td>7</td>
<td>[7]</td>
</tr>
<tr>
<td>dequeue()</td>
<td>7</td>
<td>[]</td>
</tr>
<tr>
<td>dequeue()</td>
<td>“exception”</td>
<td>[]</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>true</td>
<td>[]</td>
</tr>
<tr>
<td>enqueue(9)</td>
<td>–</td>
<td>[9]</td>
</tr>
<tr>
<td>enqueue(7)</td>
<td>–</td>
<td>[9, 7]</td>
</tr>
<tr>
<td>size()</td>
<td>2</td>
<td>[9, 7]</td>
</tr>
<tr>
<td>enqueue(3)</td>
<td>–</td>
<td>[9, 7, 3]</td>
</tr>
<tr>
<td>enqueue(5)</td>
<td>–</td>
<td>[9, 7, 3, 5]</td>
</tr>
<tr>
<td>dequeue()</td>
<td>9</td>
<td>[7, 3, 5]</td>
</tr>
</tbody>
</table>
Array-based Queue

- A simple way of implementing the Queue ADT uses an array.
  - An array of size $n$, which is the maximum number of elements in queue.
  - Insertion and deletion of elements occur at two different sides of the array.
- Two variables keep track of front and rear of the queue.
  - front ($f$): index of the front element
  - rear ($r$): index immediately past the rear element

Queue Interface in Java

```java
public interface Queue {
    public int size();
    public boolean isEmpty();
    public Object peek() throws EmptyQueueException();
    public void enqueue(Object element) throws FullQueueException();
    public Object dequeue() throws EmptyQueueException();
}
```

- Restrictions
  - The execution of `dequeue()` and `peek()` on an empty queue throws an `EmptyQueueException()`.
  - The execution of `enqueue()` on a full queue throws an `FullQueueException()`.

Queue Operations

- Implementation of a circular array: modulo operator
  - `size()` function
    - Algorithm $size()$
      - return $(N - f + r) \mod N$
      - if $f < r$ then return $r - f$
      - else return $N - (f - r)$
  - `isEmpty()` function
    - Algorithm `isEmpty()`
      - return `size()==0`

No more insertions!

Emitter and Receiver
Queue Operations

- Element insertion
  - enqueue(): add an element at rear
  - The array storing the queue elements may become full.
    - Front and back indicate the same element in the queue array.
    - An enqueue operation will then throw a FullQueueException().
    - Limitation of the array-based implementation

  Algorithm enqueue(o)
  
  if size() = N - 1 then
  throw FullQueueException()
  else
  Q[r] ← o
  r ← (r + 1) mod N

Queue Implementation in Java

```java
public class BoundedQueue implements Queue {
    private Object[] array;
    private int size = 0;
    private int front = 0; // index of the current front item
    private int rear = 0; // index of next item to be added

    public BoundedQueue(int capacity) {
        array = new Object[capacity];
    }

    public boolean isEmpty() {
        return (size == 0);
    }

    public int size() {
        return size;
    }
}
```

Queue Operations

- Element deletion
  - dequeue(): remove an element from front
  - The array storing the queue elements may become empty.
    - Front and back indicate the same element in the queue array.
    - An dequeue operation will then throw a EmptyQueueException().

  Algorithm dequeue()
  
  if isEmpty() then
  throw EmptyQueueException()
  else
  o ← Q[f]
  f ← (f + 1) mod N
  return o

Queue Implementation in Java

```java
public void enqueue(Object item) {
    if (size == array.length) throw new FullQueueException();
    array[rear] = item;
    rear = (rear + 1) % array.length;
    size++;
}

public Object dequeue() {
    if (size == 0) throw new EmptyQueueException();
    Object item = array[front];
    array[front] = null;
    front = (front + 1) % array.length;
    size--;
    return item;
}
```
### Array-based Queue

- **Performance**
  - $n$: the number of elements in the queue
  - Space complexity: $O(n)$
  - Time complexity of all operations: $O(1)$

- **Limitations**
  - The maximum size of the queue must be defined a priori and cannot be changed.
  - Trying to enqueue a new element into a full queue causes an exception.
  - Implementation with linked list may be more desirable.

### Queue Based on Linked List

```java
public class LinkedQueue implements Queue {
    private class Node {
        public Object data;
        public Node next;
        public Node(Object data, Node next) {
            this.data = data;
            this.next = next;
        }
    }
    private Node front = null;
    private Node rear = null;
    public boolean isEmpty() {
        return (front == null);
    }
    public int size() {
        int count = 0;
        for (Node node = front; node != null; node = node.next)
            count++;
        return count;
    }
    public void enqueue(Object item) {
        Node newNode = new Node(item, null);
        if (isEmpty())
            front = newNode;
        else
            rear.next = newNode;
        rear = newNode;
    }
    public Object dequeue() {
        if (isEmpty())
            throw new EmptyQueueException();
        Object item = front.data;
        if (rear == front)
            rear = null;
        front = front.next;
        return item;
    }
    public Object peek() {
        if (front == null)
            throw new EmptyQueueException();
        return front.data;
    }
}
```
Applications of Queue

- Direct applications
  - Waiting lists
  - Access to shared resources (e.g., printer)
  - Multiprogramming
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue Q by repeatedly performing the following steps:
  - e = Q.dequeue();
  - doService(e);
  - Q.enqueue(e);