COMP 250

Lecture 8

queue ADT

Sept. 23, 2016
Queue

Queues are heavily used in OS (operating systems) e.g. process scheduling.
Stack

push(e)

pop()

LIFO  
(last in,  
first out)

Queue

enqueue(e)

decoquettue()  

FIFO  
(first in,  
first out)
ADT’s (abstract data types)

• List
  add(i,e), remove(i), get(i), set(i), .....

• Stack
  push, pop(), ..

• Queue
  enqueue(e), dequeue()

Although stacks and queues consist of a finite ordered set of elements, strictly speaking, they are not lists since their operations do not allow one to index directly to the arbitrary elements.
Queue Example

enqueue(a)
enqueue(b)
dequeue()
enqueue(c)
enqueue(d)
enqueue(e)
dequeue()
enqueue(f)
dequeue()
enqueue(g)
Implementing a queue with a singly linked list.

enqueue( ) = addLast( )

dehqueue( ) = removeFirst( )
Implementing a queue with an array list. (1st attempt)

length = 4

```
enqueue(a)
enqueue(b)
dequeue()
enqueue(c)
enqueue(d)
enqueue(e)
dequeue()
enqueue(f)
dequeue()
enqueue(g)
```

Indices:
```
0 1 2 3
```
Implementing a queue with an array. (2\textsuperscript{nd} attempt)

Use head and tail indices 
(tail = head + size – 1)

<table>
<thead>
<tr>
<th>Operation</th>
<th>State</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>enqueue(a)</td>
<td>a---</td>
<td>(0, 0)</td>
</tr>
<tr>
<td>enqueue(b)</td>
<td>ab--</td>
<td>(0, 1)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>-b--</td>
<td>(1, 1)</td>
</tr>
<tr>
<td>enqueue(c)</td>
<td>-bc-</td>
<td>(1, 2)</td>
</tr>
<tr>
<td>enqueue(d)</td>
<td>-bcd</td>
<td>(1, 3)</td>
</tr>
<tr>
<td>enqueue(e)</td>
<td>-bcde---</td>
<td>(1, 4)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>--cde---</td>
<td>(2, 4)</td>
</tr>
<tr>
<td>enqueue(f)</td>
<td>--cdef--</td>
<td>(2, 5)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>---def--</td>
<td>(3, 5)</td>
</tr>
<tr>
<td>enqueue(g)</td>
<td>---defg--</td>
<td>(3, 6)</td>
</tr>
</tbody>
</table>

Start with length = 4.

Need to increase length of array.
Circular array

length = 4

0123

length = 8

01234567
Circular array

tail = (head + size – 1) % length
Implementing a queue with a circular array (GOOD)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Queue</th>
<th>Array</th>
<th>(Head, Tail, Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>enqueue(a)</td>
<td>a</td>
<td>a-----</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>enqueue(b)</td>
<td>ab</td>
<td>ab--</td>
<td>(0, 1, 2)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>b</td>
<td>-b--</td>
<td>(1, 1, 1)</td>
</tr>
<tr>
<td>enqueue(c)</td>
<td>bc</td>
<td>-bc-</td>
<td>(1, 2, 2)</td>
</tr>
<tr>
<td>enqueue(d)</td>
<td>bcd</td>
<td>-bcd</td>
<td>(1, 3, 3)</td>
</tr>
<tr>
<td>enqueue(e)</td>
<td>bcde</td>
<td>ebcd</td>
<td>(1, 0, 4)</td>
</tr>
</tbody>
</table>

\[ \text{tail} = (\text{head} + \text{size} - 1) \% \text{length} \]
The code below does not properly handle the case that size == 1. See lecture notes where this has been corrected. Note that, when size == 0, head is different from tail. Also, when queue is initialized, head == 0 and tail == length – 1.

decompose( ){
    // check that size >=1 (omitted)
    element = queue[ head ]
    if (size > 1)
        head = (head + 1) % length
    size = size – 1
    // don’t adjust tail
    return element
}
How to enqueue if the array is full?

enqueue( element ){
    if ( size == length)
        increase length of array and rearrange
        size = size + 1
        tail = (tail + 1) % length
        queue[tail] = element
}
The example shown in the following slide is slightly different from the one used in the lecture.

Please see lecture notes for further discussion of enqueueing an element when the array is full.
increase length of array and rearrange

WHY?
enqueue( element ){
    if ( size == length) { 
        // increase length of array

        create a bigger array called tmp
        for i = 0 to queue.length - 1
            tmp[i] = queue[ (head + i) % queue.length ]
        head = 0
        tail = size - 1
    }
    size = size + 1
    tail = (tail + 1) % length
    queue[tail] = element
}
Exercise: Use stack(s) to implement a queue.

```plaintext
enqueue( e ){  // add element
    :
}

dequeue() {  // remove ‘oldest’ element
    :
}
```

Write pseudocode for these two methods that uses a stack, namely use the operations `push(e)`, `pop()`, `isEmpty()`.
while ( ! s.isEmpty() ){
  tmpS.push( s.pop() )
}

S    tmpS

s    tmpS
Some possibly confusing terminology
(ADT, Java API, Java interface)

• **List** interface
  add(i,e), remove(i), get(i), set(i), .....  

• **Stack** class
  push, pop(), ..

• **Queue** interface
  offer( e ), poll (), ....