Tutorial 5

Queue ADT and Circular Arrays

Lecturer: Dr Andrew Hines *TA: Esri Ni*

5.1 Queue ADT

5.1.1 Definition

- A queue's insertion and removal routines follow the first-in-first-out (FIFO) principle.
- Elements may be inserted at any time, but only the element which has been in the queue the longest may be removed.
- Elements are inserted at the rear (enqueued) and removed from the front (dequeued)

5.1.2 The Queue Abstract Data Type

The queue supports two fundamental methods:

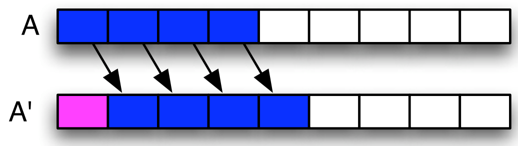
- enqueue(o): Insert object o at the rear of the queue.
  Input: Object; Output: none
- dequeue(): Remove the object from the front of the queue and return it; an error occurs if the queue is empty.
  Input: none; Output: Object

These support methods should also be defined:

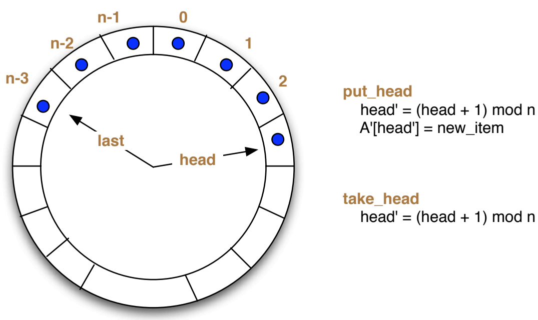
- size(): Return the number of objects in the queue.
  Input: none; Output: integer
- is_empty(): Return a boolean value that indicates whether the queue is empty.
  Input: none; Output: boolean
- front(): Return, but do not remove, the front object in the queue; an error occurs if the queue is empty.
  Input: none; Output: Object

5.1.3 Circular Arrays Recap

Adding an element at the start of a standard array means we need to move the rest of the elements.



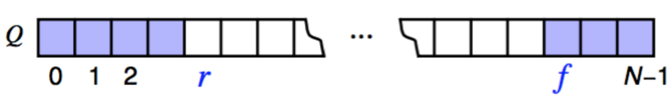
By joining the beginning and end in a circle, the capacity is available at either end of the array and we just move the address of the start of the array.



5.1.4 A Circular Array-Based Queue

- Create a queue using an array in a circular fashion with a size N
- The queue consists of an N-element array Q and two integer variables:
  - f, index of the front element
  - r, index of the element after the rear one
- "normal configuration"

- "wrapped around" configuration



- what does $f = r$ mean?

5.1.5 Exercise – Pseudo-code for Circular Array-based Queue

Develop the pseudo-code for a circular array-based "implementation" of the Queue ADT operations listed above in section 5.1.2. Create a queue using an array by specifying a maximum size n for our queue, and two integers pointers f and r for the front rank and the next available rank (rear) indexes.
Remember that the remainder operator (denoted `%` or `mod`) will allow you to loop back around. If you are confused by this, try some examples in a python console, e.g. for $n = 5$ and $f = 5$ or $f = 15$.

Algorithm 1 size

Input: A an array representing a queue, f and r two integers representing the front and the rear ranks
Output: the number of elements in the queue (-1 if empty)
  return $(n - f + r)\%n$

Algorithm 2 is empty

Input: A an array representing a queue, f and r two integers representing the front and the rear ranks
Output: $true$ if the queue is empty
  return $f = r$

Algorithm 3 front

Input: A an array representing a queue, f and r two integers representing the front and the rear ranks
Output: returns the front element
  if $is\_empty()$ **then**
    error queue empty
  end if
  return $A[f]$

Algorithm 4 dequeue

Input: A an array representing a queue, f and r two integers representing the front and the rear ranks
Output: return $elem$ the first element and remove it from the queue
  if $is\_empty()$ **then**
    error queue empty
  end if
  $tmp \leftarrow A[f]$
  $A[f] \leftarrow None$
  $f \leftarrow f + 1\%n$
  return tmp

Algorithm 5 enqueue

Input: A an array representing a queue, f and r two integers representing the front and the rear ranks, e and element
Output: add an element at the rear of the queue
  if $size() = n - 1$ **then**
    error queue full
  end if
  $A[r] \leftarrow e$
  $r \leftarrow (r + 1)\%n$

Method	Time
size	O(1)
is_empty	O(1)
first	O(1)
enqueue	O(1)
dequeue	O(1)