FIFO Queue (II): The Array-based Implementation

CSCD 300 – Data Structures

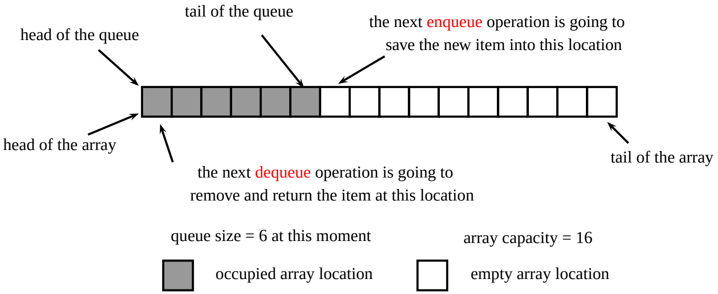
Eastern Washington University

Goal

We will demonstrate how to implement the conceptual FIFO queue data structure by using a physical array data structure.

An attempt to use array to implement the FIFO queue

- Initialize an array which is going to physically host the FIFO queue.
- The leftmost non-empty array location is the head of the queue. `Dequeue()` always removes and returns the head item, if the head item is available.
- The rightmost non-empty array location is the tail of the queue. `Enqueue(item)` always inserts the new `item` into the location right after the current tail, if such a location is available.



tail of the queue

head of the queue

the next enqueue operation is going to save the new item into this location

head of the array

the next dequeue operation is going to remove and return the item at this location

tail of the array

queue size = 6 at this moment

array capacity = 16

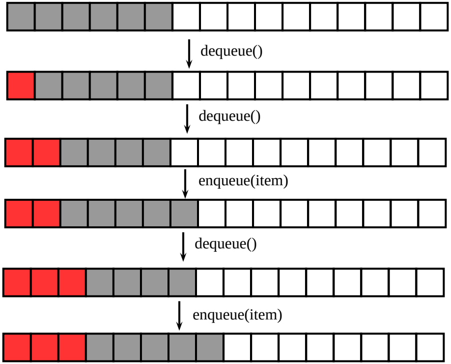
▨ occupied array location

□ empty array location

An example FIFO queue after six successive `enqueue()` operations after initialization.

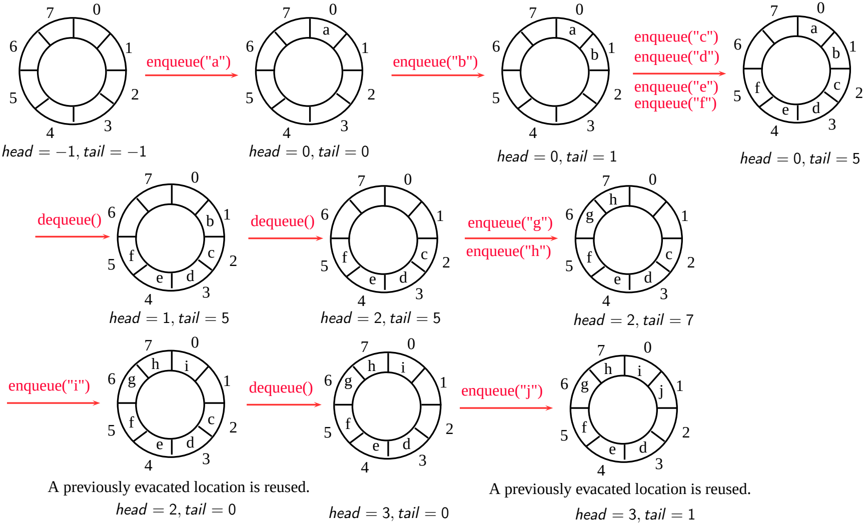
The drawback of the prior attempt

- All red locations vacated from dequeue() operations will no longer be used.
- Eventually we will have no locations to enqueue the new items, although many locations (those are in red) are still not being used.



How to fix this issue ? Recycle the vacated locations.

View the array as a ring and recycle the vacated locations



head = −1, *tail* = −1

enqueue("a")

head = 0, *tail* = 0

enqueue("b")

head = 0, *tail* = 1

enqueue("c")
enqueue("d")
enqueue("e")
enqueue("f")

head = 0, *tail* = 5

dequeue()

head = 1, *tail* = 5

dequeue()

head = 2, *tail* = 5

enqueue("g")
enqueue("h")

head = 2, *tail* = 7

enqueue("i")

A previously evacated location is reused.

head = 2, *tail* = 0

dequeue()

head = 3, *tail* = 0

enqueue("j")

A previously evacated location is reused.

head = 3, *tail* = 1

How to code this idea ? Use modulo to update the `head` and `tail`.

Pseudocode (See the attached Java code for the full implementation.)

Initialization

```
Init an array Q of capacity n;
head = tail = -1; //the index of array locations corresponding
                  //to the head and tail of the queue.
size = 0;         //#items in the queue
--
Time cost: O(1)
```

enqueue(item)

```
if(size == n) return error;
if(size == 0)
   Q[0] = item; head = tail = 0;
else
   tail = (tail+1) mod n; //!!
   Q[tail] = item;
size++;
--
Time cost: O(1)
```

dequeue()

```
if(size == 0) return error;
ret = Q[head];
if(size == 1)
   head = tail = -1;
else
   head = (head + 1) mod n; //!!
size --;
return ret;
--
Time cost: O(1)
```