Double-ended Queue (I): Introduction and the Array-based Implementation

CSCD 300 – Data Structures

Eastern Washington University

Goal

We will discuss the concept of the logical double-ended queue and its implementation using a physical array.

Outline

1. Double-ended queue (Deque)

2. The array-based implementation

Double-ended queue (Deque)

The conceptual view

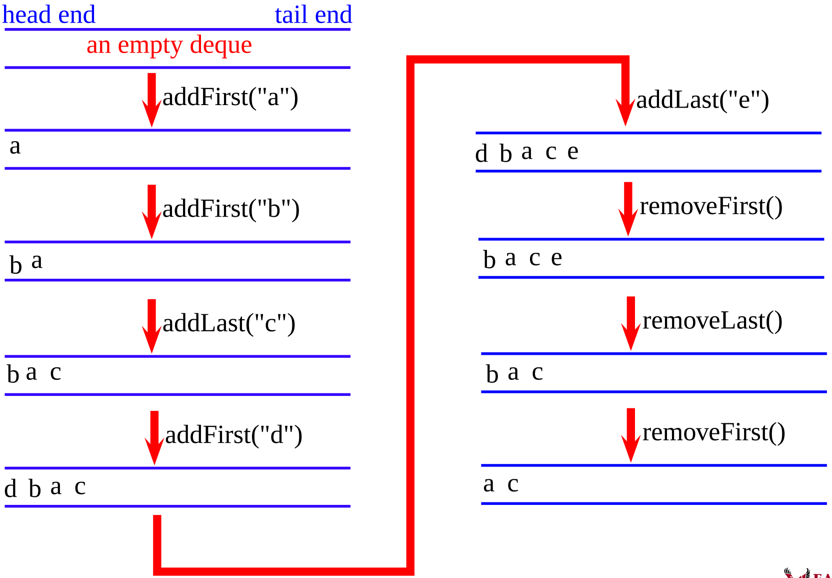
- It is still a queue, but both insert and delete operations can happen on both ends of the queue.
- To distinguish the two ends of the queue, we name one end as the head of queue and the other end as the tail of the queue.

Basic API methods

- `getFirst()`: return the head item of the deque.
- `getLast()`: return the tail item of the deque.
- `addFirst(item)`: add the `item` into the head of the deque.
- `addLast(item)`: add the `item` into the tail of the deque.
- `removeFirst()`: remove and return the head item in the deque.
- `removeLast()`: remove and return the tail item in the deque.

An example follows ...

An example sequence of deque's operations



head end tail end

an empty deque

addFirst("a")

a

addFirst("b")

b a

addLast("c")

b a c

addFirst("d")

d b a c

addLast("e")

d b a c e

removeFirst()

b a c e

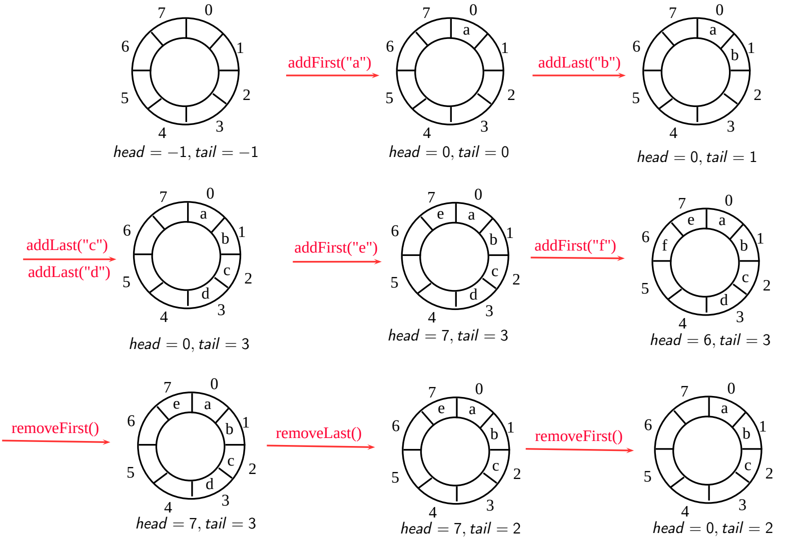
removeLast()

b a c

removeFirst()

a c

The array-based implementation: view the array as a ring



$\textit{head} = -1, \textit{tail} = -1$

addFirst("a")

$\textit{head} = 0, \textit{tail} = 0$

addLast("b")

$\textit{head} = 0, \textit{tail} = 1$

addLast("c")
addLast("d")

$\textit{head} = 0, \textit{tail} = 3$

addFirst("e")

$\textit{head} = 7, \textit{tail} = 3$

addFirst("f")

$\textit{head} = 6, \textit{tail} = 3$

removeFirst()

$\textit{head} = 7, \textit{tail} = 3$

removeLast()

$\textit{head} = 7, \textit{tail} = 2$

removeFirst()

$\textit{head} = 0, \textit{tail} = 2$

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

Initialization

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

getFirst()

```
if(size > 0) return DQ[head];
--
Time cost: O(1)
```

getLast()

```
if(size > 0) return DQ[tail];
--
Time cost: O(1)
```

continue ...

addFirst(item)

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

addLast(item)

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

removeFirst()

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

removeLast()

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