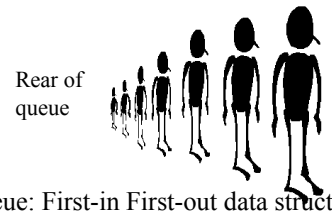
Queues, deques, and doubly-linked lists

Lecture 20

Queues



Queue: First-in First-out data structure (FIFO)

Applications: Any first-come first-serve service

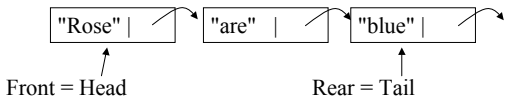
Queues operations

- void enqueue (Object o)
  - Add o to the rear of the queue
- Object dequeue()
  - Returns object at the front of the queue and removes it from the queue. Exception thrown if queue is empty.
- Object front()
  - Returns object at the front of the queue but doesn't remove it from the queue. Exception if queue empty.
- int size()
  - Returns the number of objects in the queue
- boolean isEmpty()
  - returns True is queue is empty

Example

```
Queue q = new Queue()
q.enqueue("Roses")
q.enqueue("are")
q.enqueue("red")
print q.size()
print q.front()
print q.dequeue()
dequeue()
print q.queue()
print q.isEmpty()
```

Queues with linked-lists



Queue operation	Linked-list operation	Running time
enqueue(Object o)	addLast(o)	O( )
dequeue()	removeFirst()	
front()	getFirst()	
empty()	empty()	
size()	size()	O( )

What would happen if we used instead the convention: "Front of queue = tail, Rear of queue = head" ?

Double-ended queues

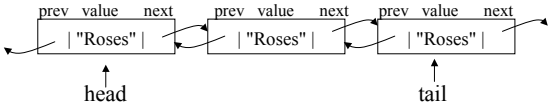
- A double-ended queue (a.k.a. "deque") allows insertions and removal from the front and back
- Deque operations with linked-lists
  - Object getFirst()
  - Object getLast()
  - addFirst(Object o)
  - addLast(Object o)
  - boolean isEmpty()
  - Object removeFirst()

  O( )
  - Object removeLast()
  - int size()

  O( )

Deques and doubly-linked-lists

- Problem: removeLast takes time O(n) with linked lists
- To do it faster, each node has to have a reference to the *previous* node in the list




```
• class node {
    node prev, next;
    Object value;
    node(Object val, node p, node n);
    node getPrev(); void SetPrev(node n);
    node getNext(); void SetNext(node n);
    Object getValue(); void setValue(Object o);
  }
```

Operations on doubly-linked-lists

```
Object removeLast() throws Exception {
    if (tail==null) throw new Exception("Empty deque");
    Object ret = tail.getValue();
    tail = tail.getPrev();
    if (tail==null) head=null;
    else tail.setNext(null);
    return ret;
}
void addFirst(Object o) {
    node n = new node(o, null, head);
    if (head != null) head.setPrev( n );
    else tail = n;
    head = n;
}
```

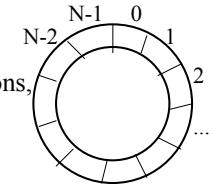
Exercise: Write all other deque methods using a doubly linked-list

Implementing deques with arrays

- Suppose we know in advance the deque will never contain more than N elements.
- We can use an array to store the elements in the deque
- Keep track of indices for head and tail

- addLast: indexTail = indexTail + 1
- addFirst: indexHead = indexHead - 1
- removeLast: indexTail = indexTail - 1
- removeFirst: indexHead = indexHead + 1

Rotating arrays



- Idea: To avoid outOfBounds exceptions, have indices "wrap around":
  (N-1) + 1 = 0
  0 - 1 = N-1
- Equivalent to arithmetic modulo N
  a mod N = rest of integer division a/N
  3 mod 7 = 3
  7 mod 7 =
  10 mod 7 =
- With a rotating array, the deque will never go out of bounds, but may overwrite itself if we try to put more than N elements into it.
- How can we check if the deque is full (has N elements?)