

DS286 | 2016-09-09

L9: Queues

Yogesh Simmhan

simmhan@cds.iisc.ac.in

Slides courtesy Venkatesh Babu, CDS

©Department of Computational and Data Science, IISc, 2016

This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/)

Copyright for external content used with attribution is retained by their original authors





Queue ADT

- FIFO Principle
- Elements **inserted only at rear** (enqueued) end and **removed from front** (dequeued)
 - Also called “Head” and “Tail”



Queue -Methods

- queue **New()** – Creates and returns an empty queue
- **Enqueue**(item *v*) – Inserts object *v* at the *rear* of the queue
- item **Dequeue()** – Removes the object from *front* of the queue. Error occurs if the queue is empty
- item **Front()** – Returns, but does not remove the front element. An error occurs if the queue is empty



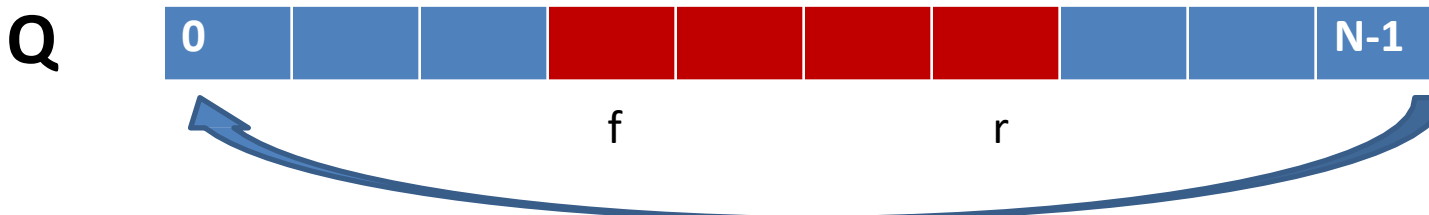
Queue – Methods & Invariants

- int **Size()** – number of items in queue
- boolean **IsEmpty()** – is size == 0
- Axioms/invariants
 - $\text{Front}(\text{Enqueue}(\text{New}(), v)) = v$
 - $\text{Dequeue}(\text{Enqueue}(\text{New}(), v)) = \text{New}()$
 - $\text{Front}(\text{Enqueue}(\text{Enqueue}(Q, w), v)) = \text{Front}(\text{Enqueue}(Q, w))$
 - $\text{Dequeue}(\text{Enqueue}(\text{Enqueue}(Q, w), v)) = \text{Enqueue}(\text{Dequeue}(\text{Enqueue}(Q, w)), v)$



Array Implementation of Queue

- Using array in *circular* fashion
 - Wraparound using mapping function (recollect from List ADT discussion)
- A max size N is specified
- Q consists of an N element array and 2 integer variables having array index:
 - f : index of the front element (head, for dequeue)
 - r : index of the element after the rear one (tail, for enqueue)



Array Implementation of Queue



- What does $f=r$ mean ?
- Resolve Ambiguity:
 - We will never add n^{th} element to Queue (declare full if the size of queue is $N-1$) .



Pseudo Code

- **int size()**
Return $(N - f + r) \bmod N$
- **bool isEmpty()**
Return $(f == r)$
- **int front()**
If `isEmpty()` then Return `QueueEmptyException`
Else Return `Q[f]`

Pseudo Code

- **int Dequeue()**

If isEmpty() then Return QueueEmptyException

$v = Q[f]$

$Q[f] = \text{null}$

$f = (f+1) \bmod N$

Return v

- **Enqueue(v)**

If size() $=n-1$ then Return QueueFullException

$Q[r] = v$

$r = (r+1) \bmod N$

Compute Complexity? Storage Complexity?



Linked List

- Problem with array: Requires the number of elements *a priori*.

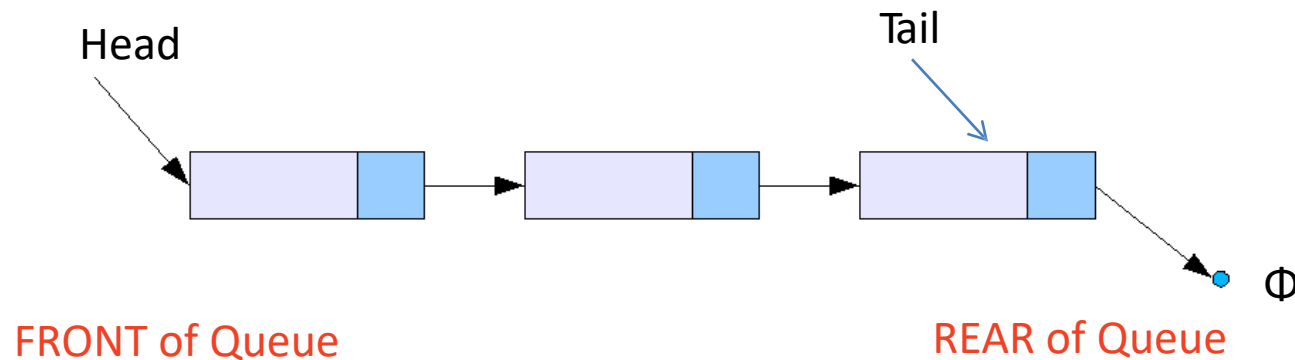




Implementation with linked List

NOTE: Different from what was mentioned in class.

Nodes (data, pointer) connected in a chain by links



- Maintain two pointers, to head and tail of linked list.
- The head of the list is FRONT of the queue, the tail of the list is REAR of the queue.
- **Why not the opposite?**



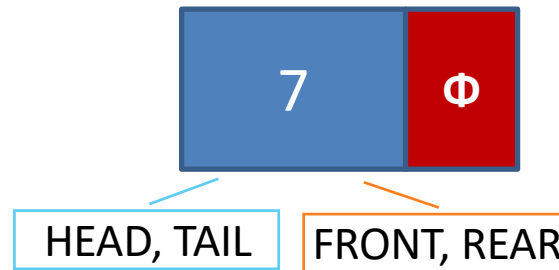
Example

Tail of linked list is REAR of queue... Enqueue at tail

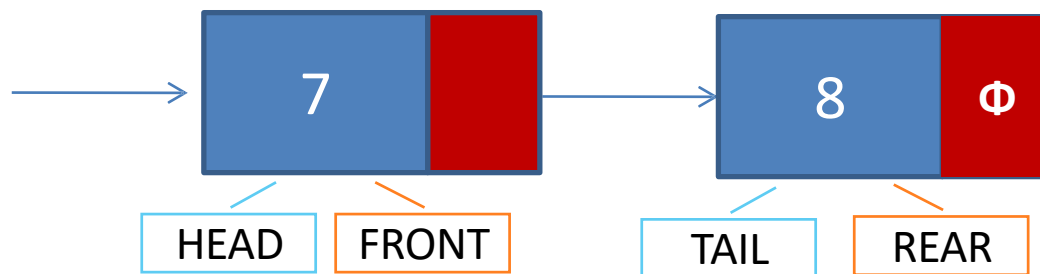
Head is FRONT of queue... Dequeue at head

TAIL = Φ , HEAD = Φ

- Enqueue 7



- Enqueue 8



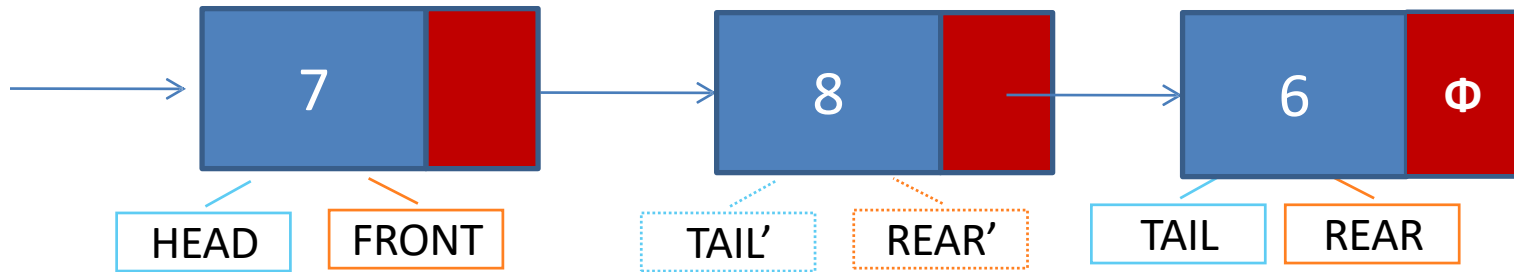


Example

Tail of linked list is REAR of queue... Enqueue at tail

Head is FRONT of queue... Dequeue at head

- Enqueue 6



```
Node n = new Node(6)
```

```
n.next = null
```

```
TAIL.next = n
```

```
TAIL = n          // NEW REAR
```

O(1) complexity to enqueue

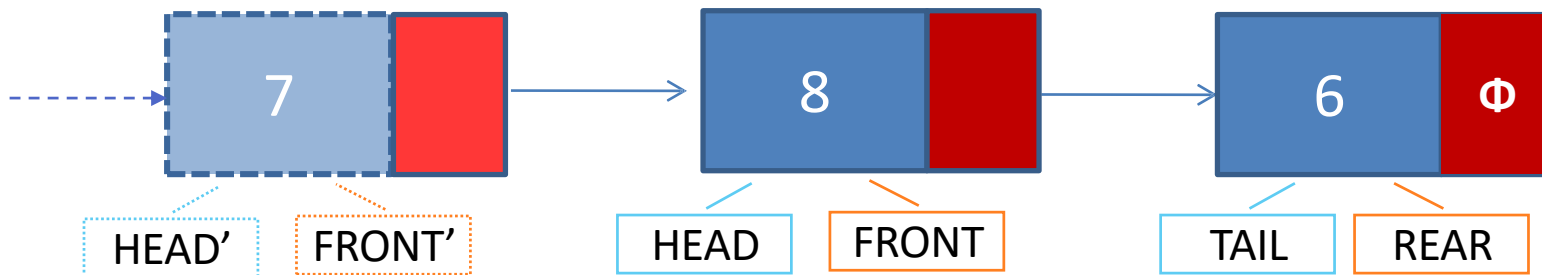


Example

Tail of linked list is REAR of queue... Enqueue at tail

Head is FRONT of queue... Dequeue at head

- Dequeue $\rightarrow 7$

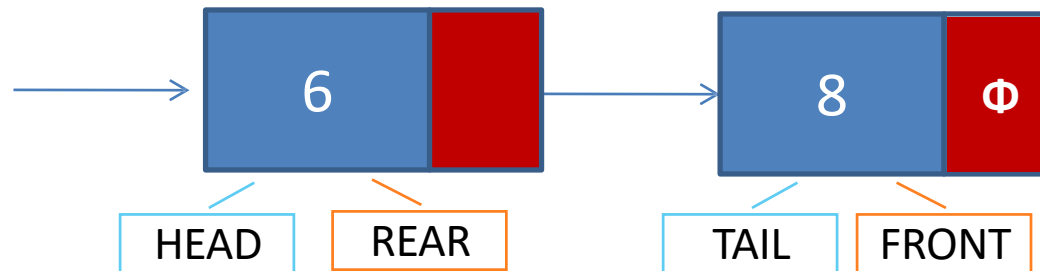


```
int v = HEAD.value
tmp = HEAD.next
delete(HEAD)
HEAD = tmp      // NEW HEAD
return v        // 7
```

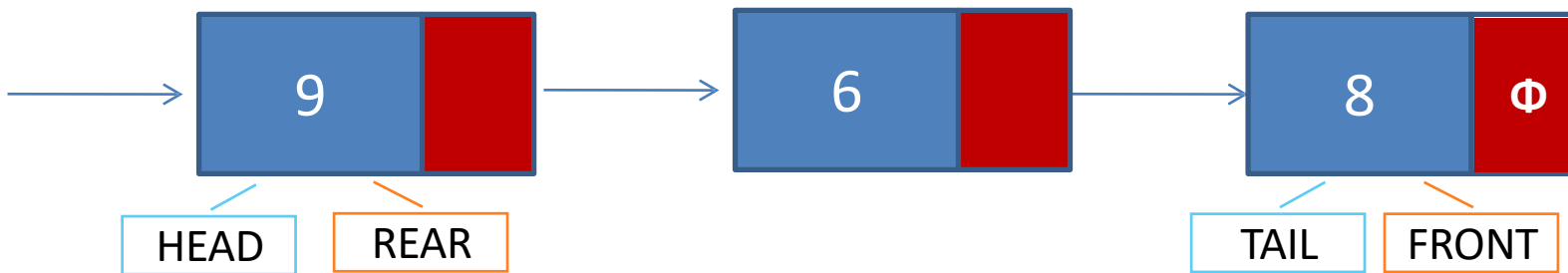
$O(1)$ complexity to dequeue

Example

What if Head of linked list is REAR of queue, Tail the FRONT?



- Enqueue 9

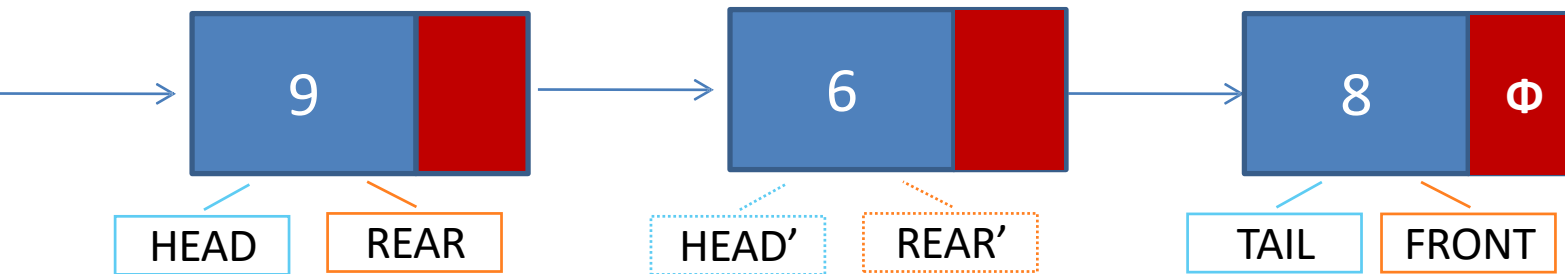




Example

What if Head of linked list is REAR of queue, Tail the FRONT?

- Enqueue 9



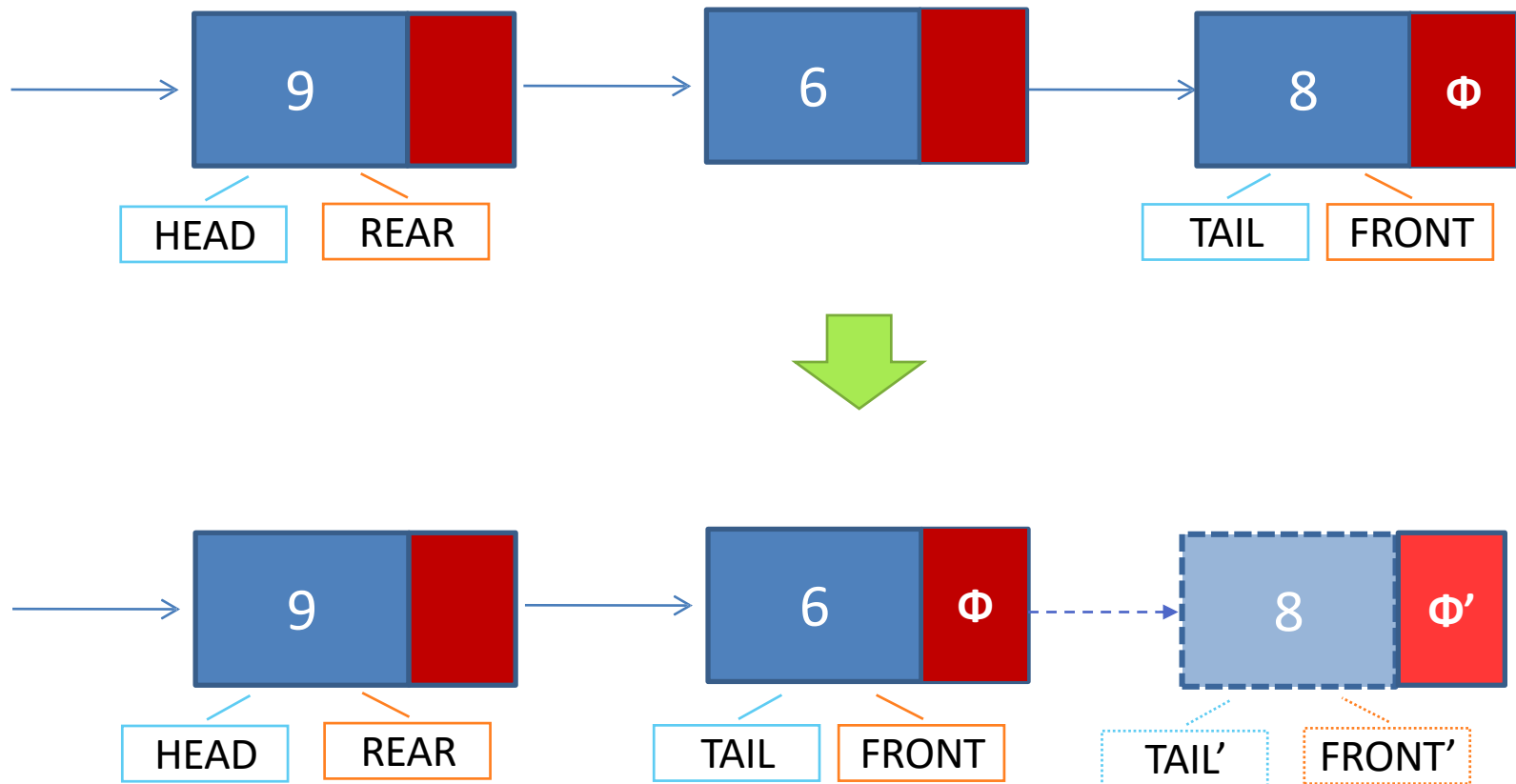
```
Node n = new Node(9)
n.next = HEAD
HEAD = n
```

O(1) complexity to enqueue

Example

What if Head of linked list is REAR of queue, Tail the FRONT?

- Dequeue $\rightarrow 8$

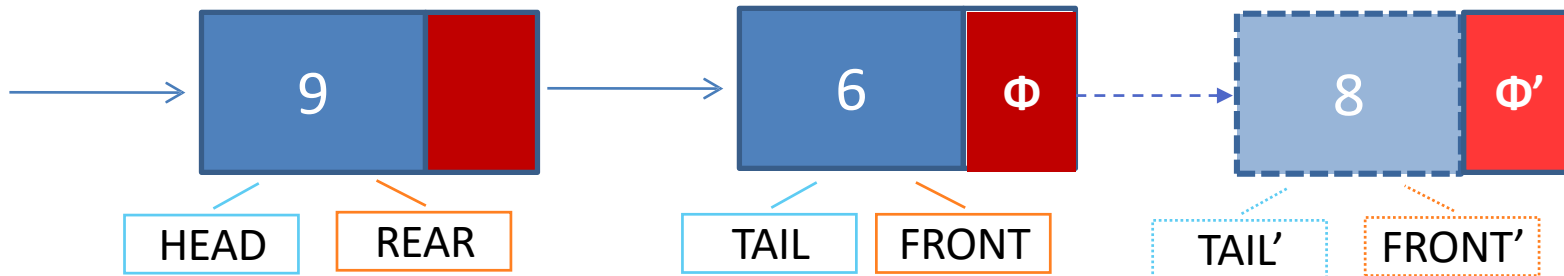


Example

What if Head of linked list is REAR of queue, Tail the FRONT?

Things don't work well!

- Dequeue $\rightarrow 8$



```
int v = TAIL.value           // 8
// tail's "previous" (6) should point to null!
n = HEAD                    // Can head be null?
while(n.next != TAIL) n = n.next;
n.next = null
delete(TAIL)
TAIL = n
return v
```

$O(N)$ complexity to dequeue



Linked List



```
typedef struct Node {  
    int value;  
    struct Node *next;    // pointer to Node  
} a_node;
```

Any linked list is a pointer to a node

```
typedef Node *list;    // head of list
```

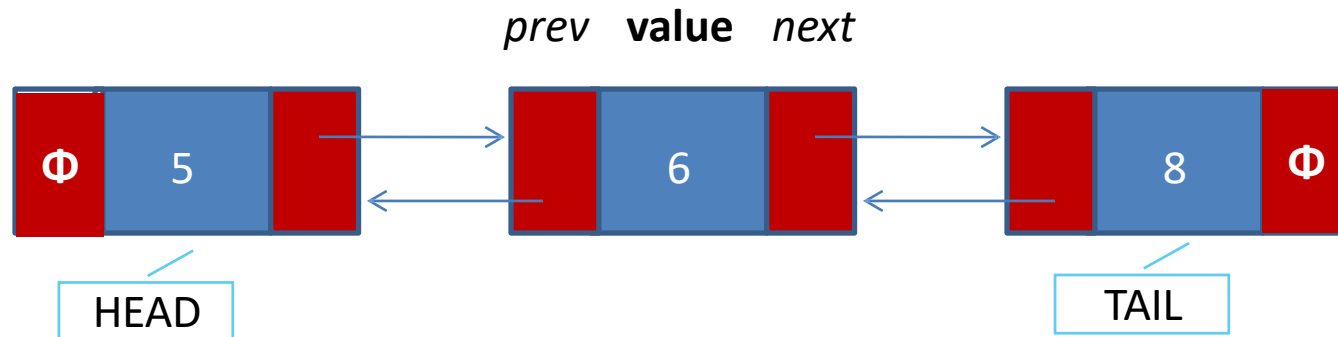


Double-Ended Queue (Deque)

- Supports insertion & deletion from front & rear
- **Supports six methods**
 - InsertFirst(item o) – Inserts 'o' at the beginning of deque
 - InsertLast(item o) – Inserts 'o' at the end of deque
 - item RemoveFirst() – removes the 1st element
 - item RemoveLast() – removes the last element
 - item First() – return first element
 - item Last() - return last element

Problem in implementing using single linked list with $O(1)$
e.g. If head is front and tail is rear, RemoveLast will require traversal from head to tail's previous

Deque as Doubly Linked List



- Nodes of doubly linked list have a **next** and a **prev** link
- All the methods of a deque using doubly linked list have a constant running time $O(1)$
 - *How?*



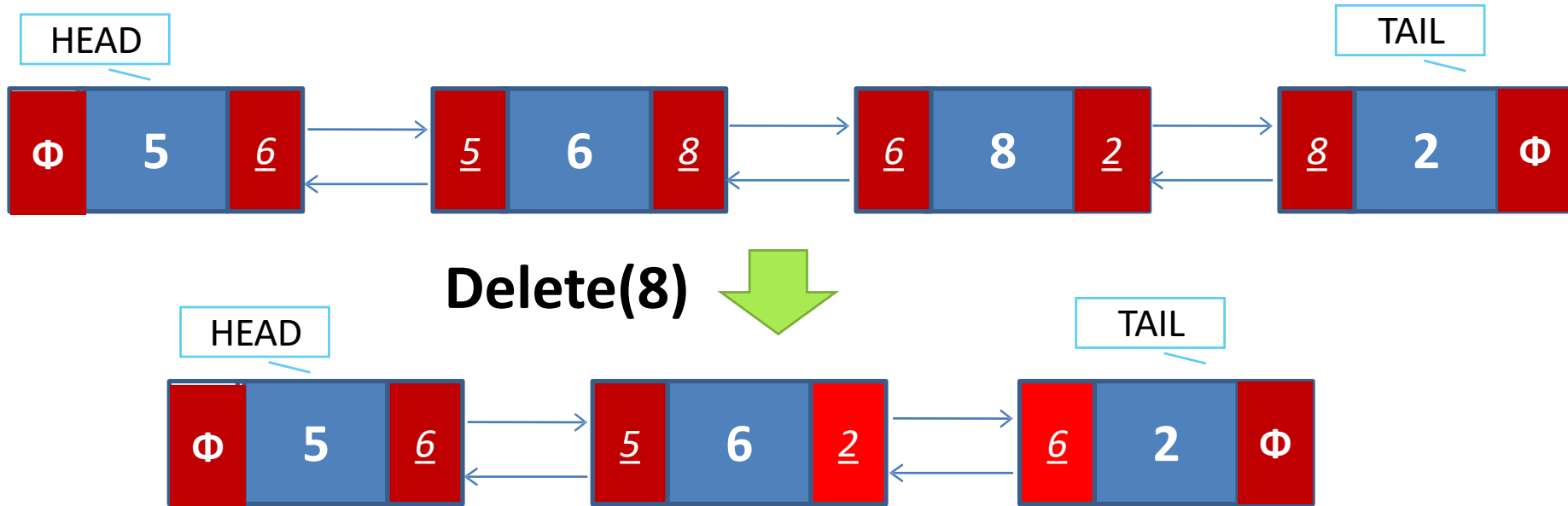
Doubly Linked List



```
typedef struct Node {  
    int value;  
    struct Node *next, *prev;  
} a_node;
```



Delete Element from Doubly Linked List



Node n = HEAD

```
while(n.next.value != v) n = n.next;
```

```
tmp = n.next
```

```
// n = 6, tmp = 8
```

```
n.next = tmp.next
```

```
// 6.next = 2
```

```
tmp.next.prev = n
```

```
// 2.prev = 6
```

```
delete(tmp)
```

```
// delete 8
```



Implement Stacks with Deques

Stack Method	Deque Method
Size()	Size()
IsEmpty()	IsEmpty()
Top()	Last()
Push()	insertLast()
Pop()	removeLast()

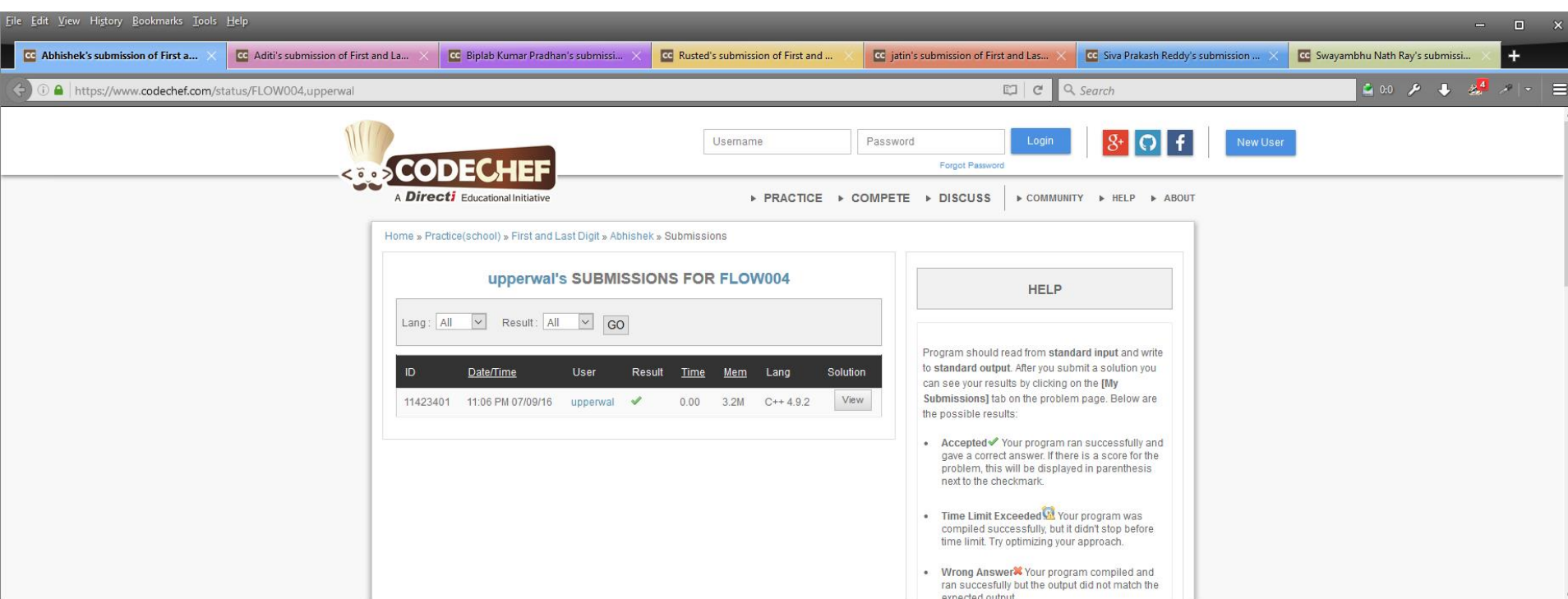


Implement Queue with Deques

Stack Method	Deque Method
Size()	Size()
IsEmpty()	IsEmpty()
front()	first()
enqueue()	insertLast()
dequeue()	removeFirst()

CodeChef: FLOW004

■ 9/Sep



The screenshot shows the CodeChef website interface. At the top, there's a navigation bar with links for PRACTICE, COMPETE, DISCUSS, COMMUNITY, HELP, and ABOUT. Below this, the user 'upperwal' is logged in. The main content area displays 'upperwal's SUBMISSIONS FOR FLOW004'. A table shows a single submission with ID 11423401, dated 11:06 PM 07/09/16, with a status of 'Accepted'. To the right, there's a 'HELP' section explaining the submission process and the meaning of different statuses: Accepted, Time Limit Exceeded, and Wrong Answer.

upperwal's SUBMISSIONS FOR FLOW004

Lang: Result:

ID	Date/Time	User	Result	Time	Mem	Lang	Solution
11423401	11:06 PM 07/09/16	upperwal	Accepted	0.00	3.2M	C++ 4.9.2	<input type="button" value="View"/>

HELP

Program should read from **standard input** and write to **standard output**. After you submit a solution you can see your results by clicking on the **[My Submissions]** tab on the problem page. Below are the possible results:

- Accepted** ✓ Your program ran successfully and gave a correct answer. If there is a score for the problem, this will be displayed in parenthesis next to the checkmark.
- Time Limit Exceeded** ⚡ Your program was compiled successfully, but it didn't stop before time limit. Try optimizing your approach.
- Wrong Answer** ✗ Your program compiled and ran successfully but the output did not match the expected output.

Tasks

- Solve sanity check problem on CodeChef by **Sep 14**
 - <https://www.codechef.com/problems/FLOW004>
- Self study (Sahni Textbook)
 - **Check:** Have you read Chapter 8 “Stacks”? Solved exercises?
 - **Read:** Chapter 9, Queues from textbook
 - **Try:** Exercise 4, 14, 18 from Chapter 9 of textbook



Questions?

©Department of Computational and Data Science, IISc, 2016

This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/)

Copyright for external content used with attribution is retained by their original authors

