

## 2-10 Elementary Data Structures

Hengfeng Wei

hfwei@nju.edu.cn

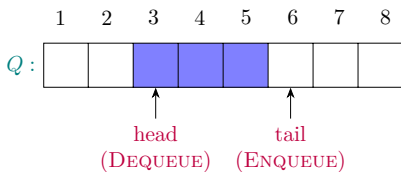
May 30, 2018

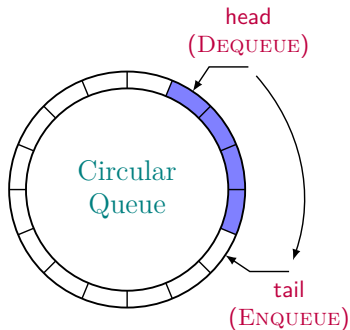
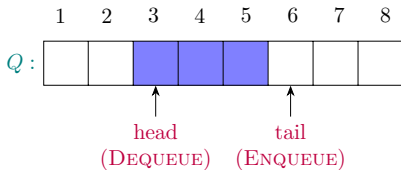


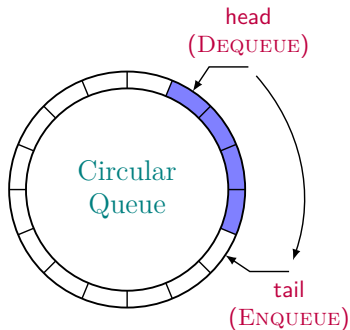
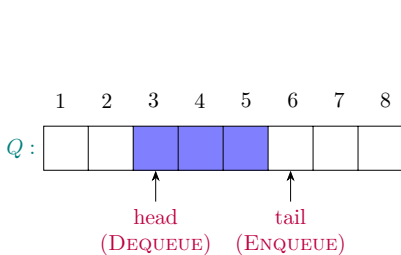






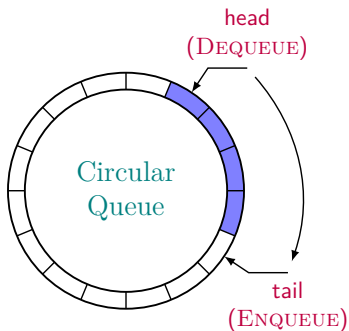






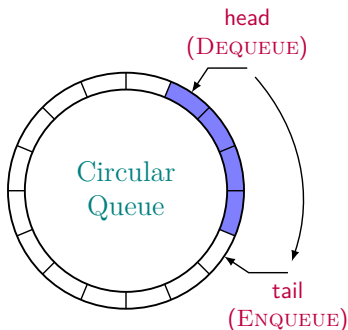
head & teal: following the same direction

## Underflow and Overflow of a Circular Queue (Problem 10.1-4)





## Underflow and Overflow of a Circular Queue (Problem 10.1-4)



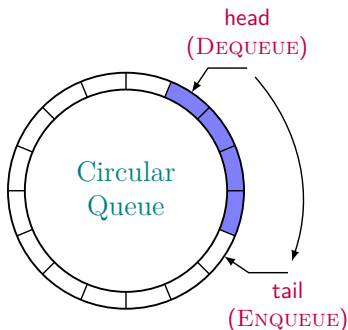
---

```
procedure DEQUEUE( $Q$ )  
  if  $Q.head = Q.tail$  then  
    return "UNDERFLOW"
```

---

...

## Underflow and Overflow of a Circular Queue (Problem 10.1-4)



---

```
procedure DEQUEUE(Q)
  if  $Q.head = Q.tail$  then
    return "UNDERFLOW"
```

---

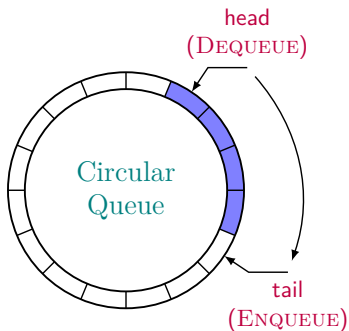
...

---

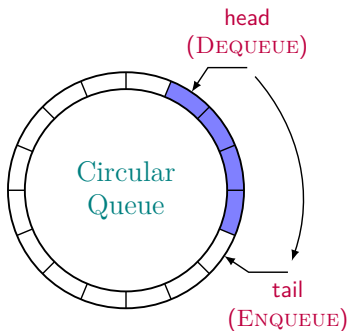
```
procedure ENQUEUE(Q, x)
  if  $Q.head = Q.tail + 1$  then
    return "OVERFLOW"
```

---

...



反馈: tail 为什么指向最后一个元素的后面? 这个太难受了。



反馈: tail 为什么指向最后一个元素的后面? 这个太难受了。

QUEUE-EMPTY

$[l, r)$     $(l, r]$     $[l, r]$     $(l, r)$

$[l, r)$   $(l, r]$   $[l, r]$   $(l, r)$

EWD831-0

Why numbering should start at zero [EWD831.html](http://EWD831.html)

To denote the subsequence of natural numbers 2, 3, ..., 12 without the pernicious three dots, four conventions are open to us:

- a)  $2 \leq i < 13$
- b)  $1 < i \leq 12$
- c)  $2 \leq i \leq 12$
- d)  $1 < i < 13$



## Why Numbering Should Start at Zero

## A Queue, Two Stacks (Problem 10.1-6)

Show how to implement a queue using two stacks.

## A Queue, Two Stacks (Problem 10.1-6)

Show how to implement a queue using two stacks.

---

**procedure** ENQUEUE( $x$ )

*Push*( $S_1, x$ )

**procedure** DEQUEUE()

**if**  $S_2 = \emptyset$  **then**

**while**  $S_1 \neq \emptyset$  **do**

*Push*( $S_2, \text{Pop}(S_1)$ )

*Pop*( $S_2$ )

---



## A Queue, Two Stacks (Problem 10.1-6)

Show how to implement a queue using two stacks.

---

```
procedure ENQUEUE( $x$ )  
    Push( $S_1, x$ )
```

Correctness?

```
procedure DEQUEUE()  
    if  $S_2 = \emptyset$  then  
        while  $S_1 \neq \emptyset$  do  
            Push( $S_2, \text{Pop}(S_1)$ )  
    Pop( $S_2$ )
```

---

## A Queue, Two Stacks (Problem 10.1-6)

Show how to implement a queue using two stacks.

---

```
procedure ENQUEUE( $x$ )  
     $Push(S_1, x)$ 
```

Correctness?

```
procedure DEQUEUE()  
    if  $S_2 = \emptyset$  then  
        while  $S_1 \neq \emptyset$  do  
             $Push(S_2, Pop(S_1))$   
     $Pop(S_2)$ 
```

---

$$\begin{aligned} &ENQ(x, t_1), ENQ(y, t_2) \wedge t_1 < t_2 \\ &\implies \\ &DEQ(x, t_3), DEQ(y, t_4) \wedge t_3 < t_4 \end{aligned}$$

## A Queue, Two Stacks (Problem 10.1-6)

Show how to implement a queue using two stacks.

Analyze the running time of the queue operations.

---

**procedure** ENQUEUE( $x$ )

*Push*( $S_1, x$ )

Correctness?

**procedure** DEQUEUE()

**if**  $S_2 = \emptyset$  **then**

**while**  $S_1 \neq \emptyset$  **do**

*Push*( $S_2, \text{Pop}(S_1)$ )

*Pop*( $S_2$ )

---

$\text{ENQ}(x, t_1), \text{ENQ}(y, t_2) \wedge t_1 < t_2$

$\implies$

$\text{DEQ}(x, t_3), \text{DEQ}(y, t_4) \wedge t_3 < t_4$

<i>item</i>	Push into $S_1$	Pop from $S_1$	Push into $S_2$	Pop from $S_2$
$x$	1	1	1	1

<i>item</i>	Push into $S_1$	Pop from $S_1$	Push into $S_2$	Pop from $S_2$
$x$	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 4$$

$$\hat{c}_{\text{DEQ}} = 0$$

<i>item</i>	Push into $S_1$	Pop from $S_1$	Push into $S_2$	Pop from $S_2$
$x$	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 4$$

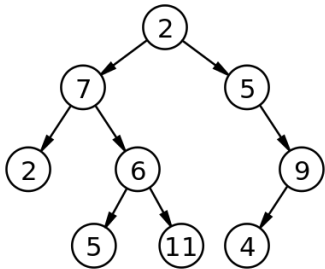
$$\hat{c}_{\text{DEQ}} = 0$$

$$\hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 1$$

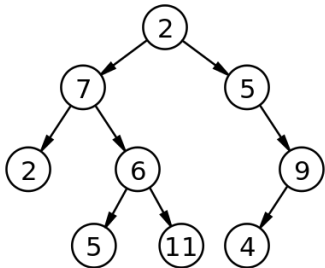
## Recursive Binary Tree Traversal (Problem 10.4 – 2)

$O(n)$



## Recursive Binary Tree Traversal (Problem 10.4 – 2)

$O(n)$



---

```
procedure RECURSIVE-DFS(t)  
  print t.key
```

```
  if t.left  $\neq$  NIL then  
    RECURSIVE-DFS(t.left)  
  if t.right  $\neq$  NIL then  
    RECURSIVE-DFS(t.right)
```

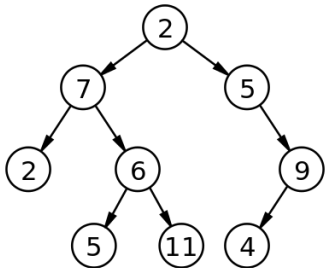
```
RECURSIVE-DFS(T.root)
```

---



## Recursive Binary Tree Traversal (Problem 10.4 – 2)

$O(n)$



---

---

```
procedure RECURSIVE-DFS(t)
```

```
  print t.key
```

```
  if t.left  $\neq$  NIL then
```

```
    RECURSIVE-DFS(t.left)
```

```
  if t.right  $\neq$  NIL then
```

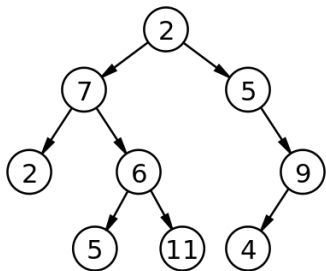
```
    RECURSIVE-DFS(t.right)
```

```
RECURSIVE-DFS(T.root)
```

---

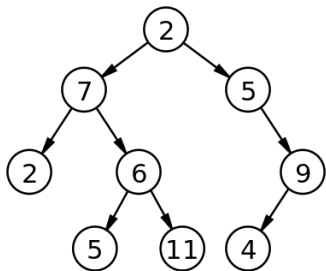
## Non-recursive Binary Tree Traversal (Problem 10.4 – 2)

$O(n)$



## Non-recursive Binary Tree Traversal (Problem 10.4 – 2)

$O(n)$



---

**procedure** ITERATIVE-DFS( $t$ ) $S.PUSH(t)$  $\triangleright S : \text{stack}$ **while**  $S \neq \emptyset$  **do** $v \leftarrow S.POP()$ **print**  $v.key$ **if**  $v.right \neq \text{NIL}$  **then** $S.PUSH(v.right)$ **if**  $v.left \neq \text{NIL}$  **then** $S.PUSH(v.left)$  $\text{ITERATIVE-DFS}(T.root)$

## “LCRS” Tree Traversal (Problem 10.4 – 2)

Thank  
You!



Office 302

Mailbox: H016

hfwei@nju.edu.cn