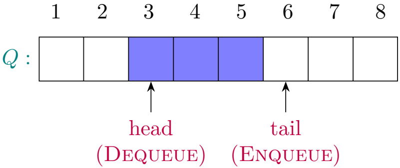
2-10 Elementary Data Structures

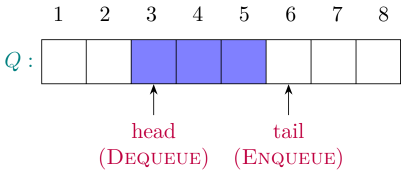
Hengfeng Wei

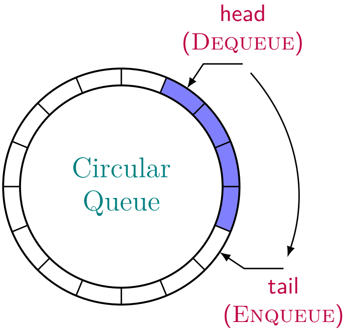
hfwei@nju.edu.cn

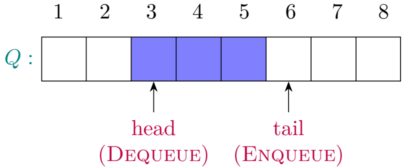
May 30, 2018

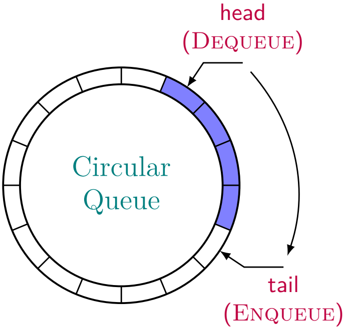

SO
EASY


1 2 3 4 5 6 7 8
Q :
head
(DEQUEUE)
tail
(ENQUEUE)


1 2 3 4 5 6 7 8
Q :
head
(DEQUEUE)
tail
(ENQUEUE)

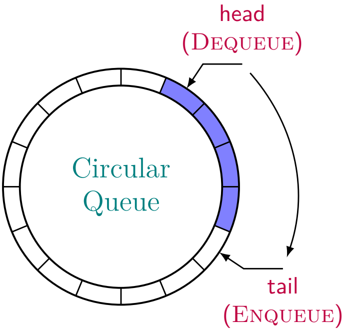

head
(DEQUEUE)
Circular
Queue
tail
(ENQUEUE)



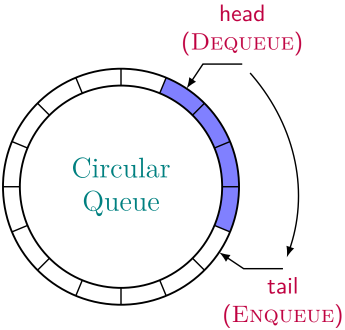


head & teal: following the same direction

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

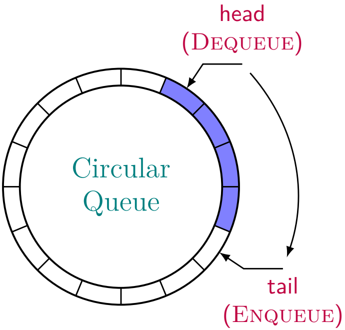


Underflow and Overflow of a Circular Queue (Problem 10.1-4)



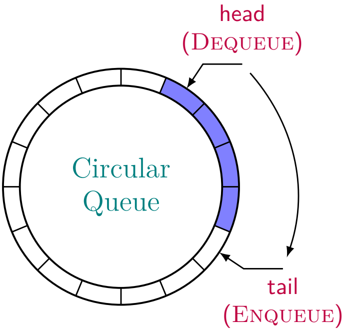
procedure $\textsc{Dequeue}(Q)$
  if $Q.head = Q.tail$ **then**
    return "UNDERFLOW"
  $\ldots$

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

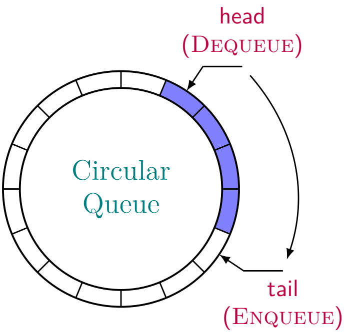


procedure Dequeue(Q)
  if $Q.head = Q.tail$ **then**
    return "UNDERFLOW"
  ...

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

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

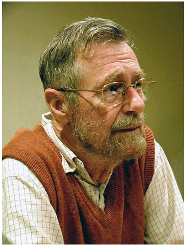
$[l, r) \quad (l, r] \quad [l, r] \quad (l, r)$

EWD831-0

Why numbering should start at zero   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 $\textsc{Enqueue}(x)$
  Push(S_1, x)

procedure $\textsc{Dequeue}()$
  if $S_2 = \emptyset$ **then**
    while $S_1 \neq \emptyset$ **do**
      Push$\Big(S_2,$ *Pop*$(S_1)\Big)$
  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)

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

Correctness?

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$\Big(S_2,$ *Pop*$(S_1)\Big)$
  Pop(S_2)

Correctness?

$$\text{Enq}(x, t_1), \text{Enq}(y, t_2) \land t_1 < t_2$$

$$\Longrightarrow$$

$$\text{Deq}(x, t_3), \text{Deq}(y, t_4) \land t_3 < t_4$$

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 $\textsc{Enqueue}(x)$
  Push(S_1, x)

procedure $\textsc{Dequeue}()$
  if $S_2 = \emptyset$ **then**
    while $S_1 \neq \emptyset$ **do**
      Push$\Big(S_2,$ *Pop*$(S_1)\Big)$
  Pop(S_2)

Correctness?

$$\textsc{Enq}(x, t_1), \textsc{Enq}(y, t_2) \land t_1 < t_2$$
$$\Longrightarrow$$
$$\textsc{Deq}(x, t_3), \textsc{Deq}(y, t_4) \land t_3 < t_4$$

item	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
x	1	1	1	1

item	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
x	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 4$$
$$\hat{c}_{\text{DEQ}} = 0$$

item	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
x	1	1	1	1

$$\hat{c}_{\text{ENQ}} = 4 \qquad \hat{c}_{\text{ENQ}} = 3$$

$$\hat{c}_{\text{DEQ}} = 0 \qquad \hat{c}_{\text{DEQ}} = 1$$

Compactify-List (Problem $10.3 - 5$)

$$\textsc{Compactify-List}(L, F)$$

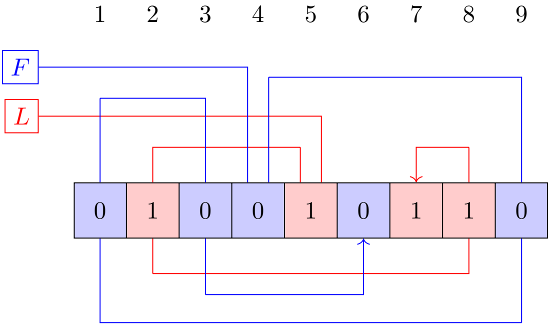
$$L: \text{ doubly linked list}, \quad |L| = n$$

$$F: \text{ doubly linked free list}, \quad |F| = m - n$$

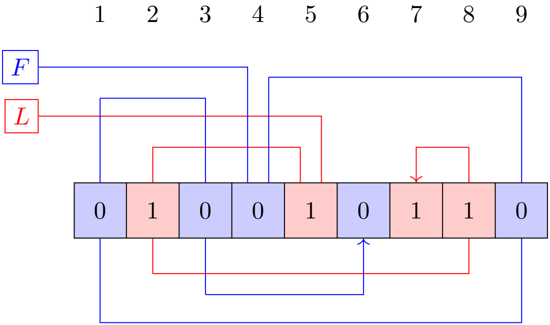
$$\Theta(n)$$

1	2	3	4	5	6	7	8	9
1	1	1	1	0	0	0	0	0

1	2	3	4	5	6	7	8	9
1	1	1	1	0	0	0	0	0

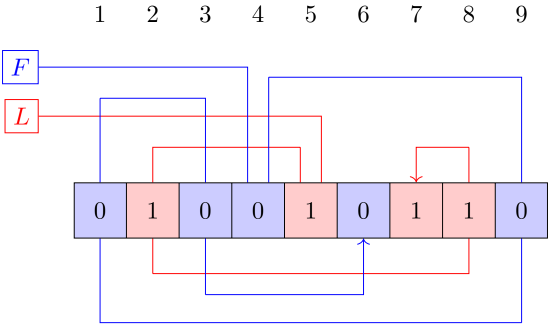


1	2	3	4	5	6	7	8	9
1	1	1	1	0	0	0	0	0



Swap $(0, 1)$ pairs

1	2	3	4	5	6	7	8	9
1	1	1	1	0	0	0	0	0



Swap $(0, 1)$ pairs

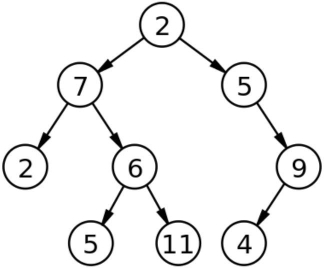
$\Theta(n)$

Hoare-Partition

11	1	20	5	12	5	16	21	9	13	17
	↑									↑
	i									j

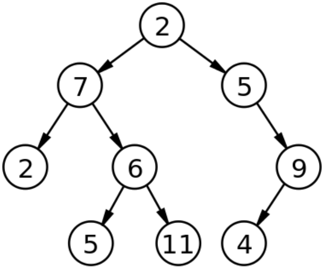
Recursive Binary Tree Traversal (Problem $10.4-2$)

$O(n)$



Recursive Binary Tree Traversal (Problem $10.4-2$)

$O(n)$



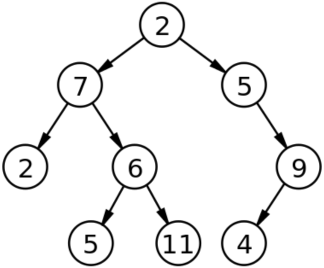
procedure $\textsc{Recursive-DFS}(t)$
  print $t.key$

  if $t.left \neq$ NIL **then**
    $\textsc{Recursive-DFS}(t.left)$
  if $t.right \neq$ NIL **then**
    $\textsc{Recursive-DFS}(t.right)$

$\textsc{Recursive-DFS}(T.root)$

Recursive Binary Tree Traversal (Problem $10.4-2$)

$O(n)$



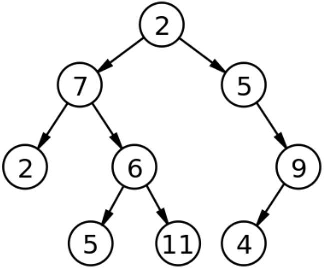
procedure $\textsc{Recursive-DFS}(t)$
    print $t.key$

    if $t.left \neq$ NIL **then**
        $\textsc{Recursive-DFS}(t.left)$
    if $t.right \neq$ NIL **then**
        $\textsc{Recursive-DFS}(t.right)$

$\textsc{Recursive-DFS}(T.root)$

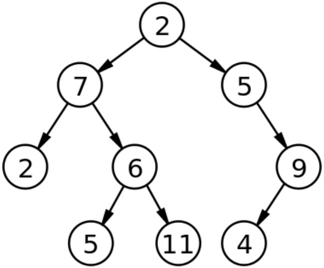
Non-recursive Binary Tree Traversal (Problem $10.4-2$)

$$O(n)$$



Non-recursive Binary Tree Traversal (Problem $10.4-2$)

$O(n)$



procedure $\text{ITERATIVE-DFS}(t)$

$S.\text{PUSH}(t)$ ▷ S : stack

while $S \neq \emptyset$ **do**

$v \leftarrow S.\text{POP}()$

print $v.key$

if $v.right \neq$ NIL **then**

$S.\text{PUSH}(v.right)$

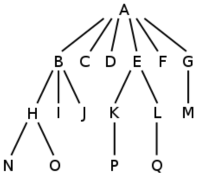
if $v.left \neq$ NIL **then**

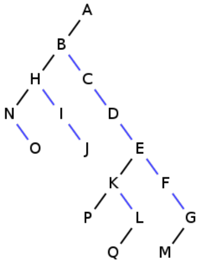
$S.\text{PUSH}(v.left)$

$\text{ITERATIVE-DFS}(T.root)$

“LCRS” Tree Traversal (Problem $10.4-2$)

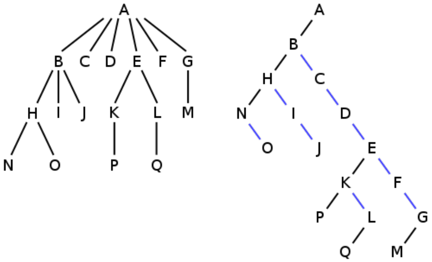
$$O(n)$$





"LCRS" Tree Traversal (Problem $10.4 - 2$)

$O(n)$



procedure $\textsc{Recursive-DFS}(t)$
  print $t.key$

  if $t.lc \neq$ NIL **then**
    $\textsc{Recursive-DFS}(t.lc)$
  if $t.rs \neq$ NIL **then**
    $\textsc{Recursive-DFS}(t.rs)$

$\textsc{Recursive-DFS}(T.root)$

Thank You!

Your opinon Matters

Office 302

Mailbox: H016

hfwei@nju.edu.cn