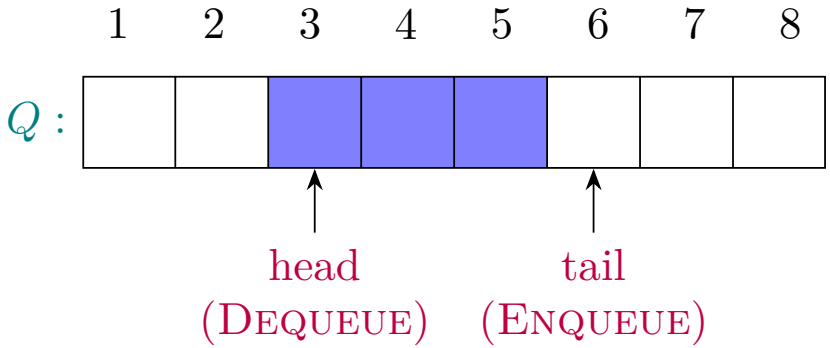
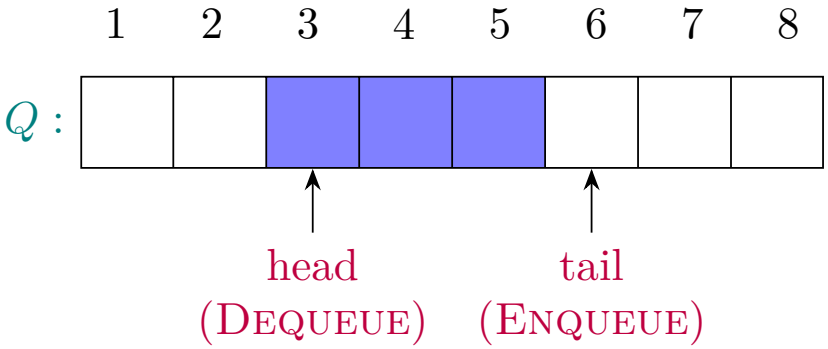
2-10 Elementary Data Structures

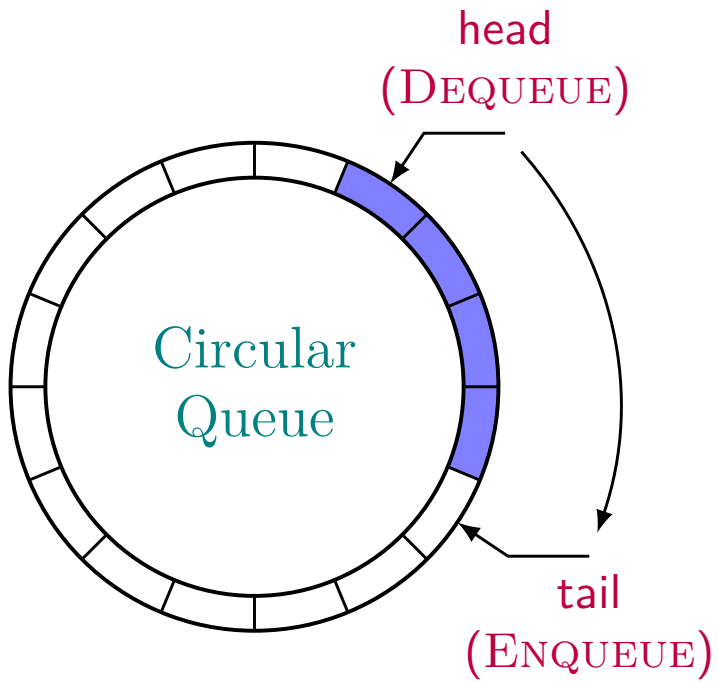
Hengfeng Wei

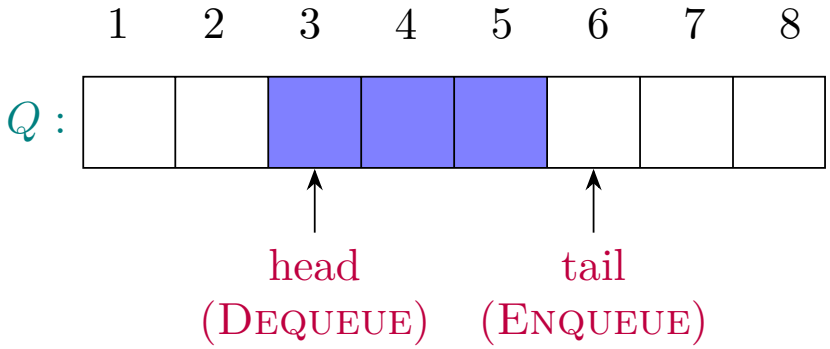
hfwei@nju.edu.cn

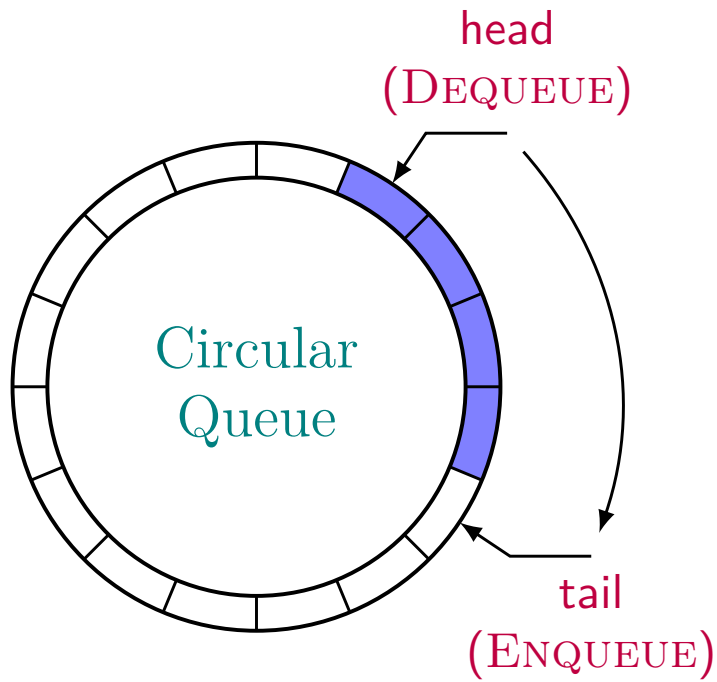
May 30, 2018


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)

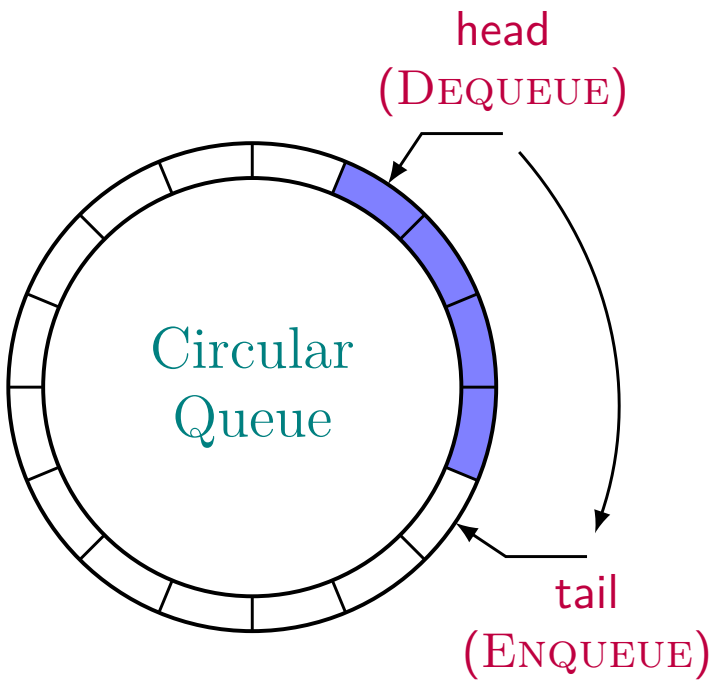

head
(DEQUEUE)
Circular
Queue
tail
(ENQUEUE)



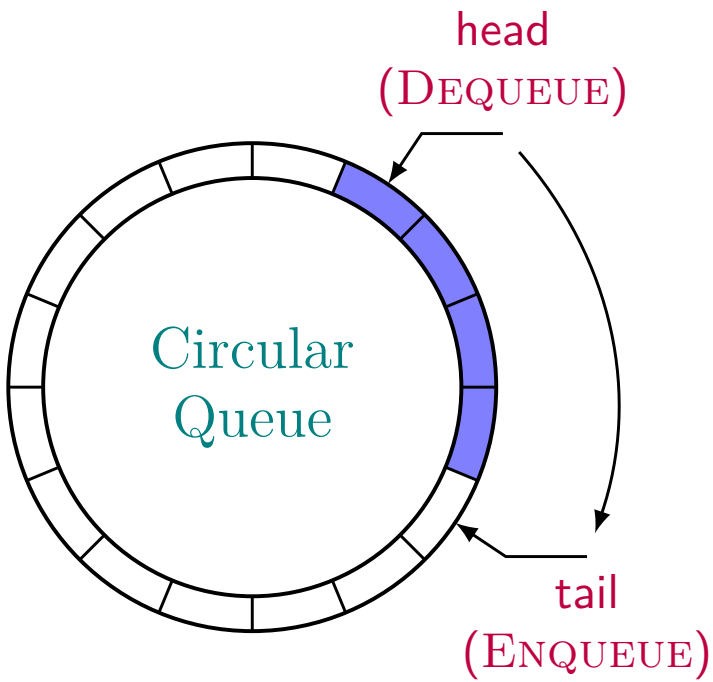


head & teal: following the same direction

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



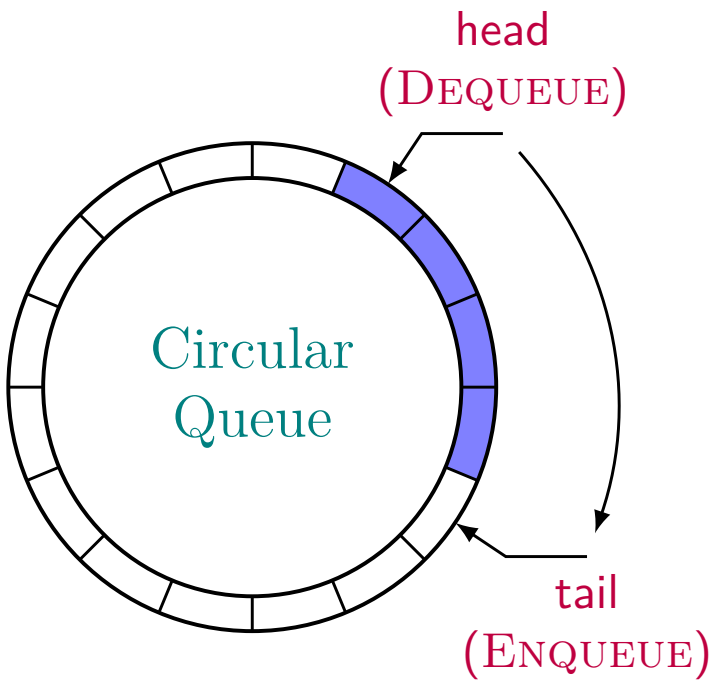
Underflow and Overflow of a Circular Queue (Problem 10.1-4)



---

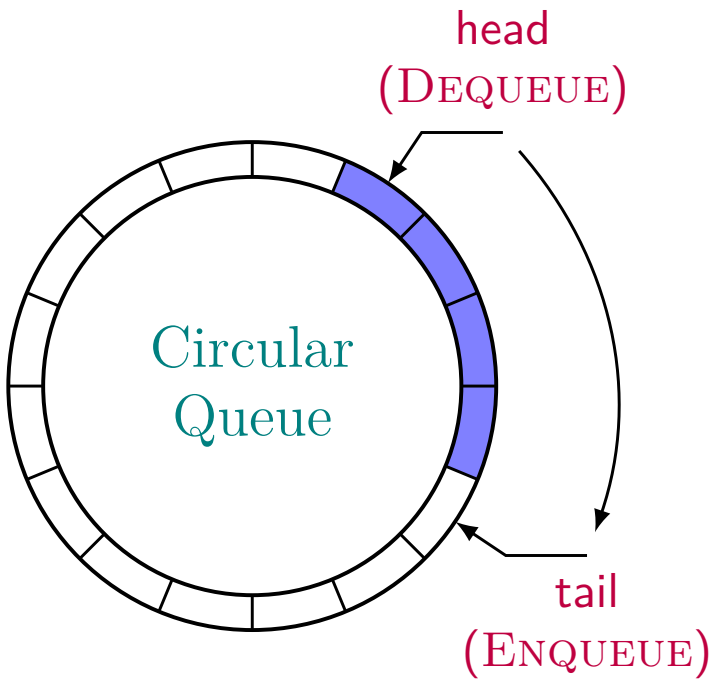
procedure DEQUEUE(Q)
  if $Q.head = Q.tail$ **then**
    return "UNDERFLOW"
  ...

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

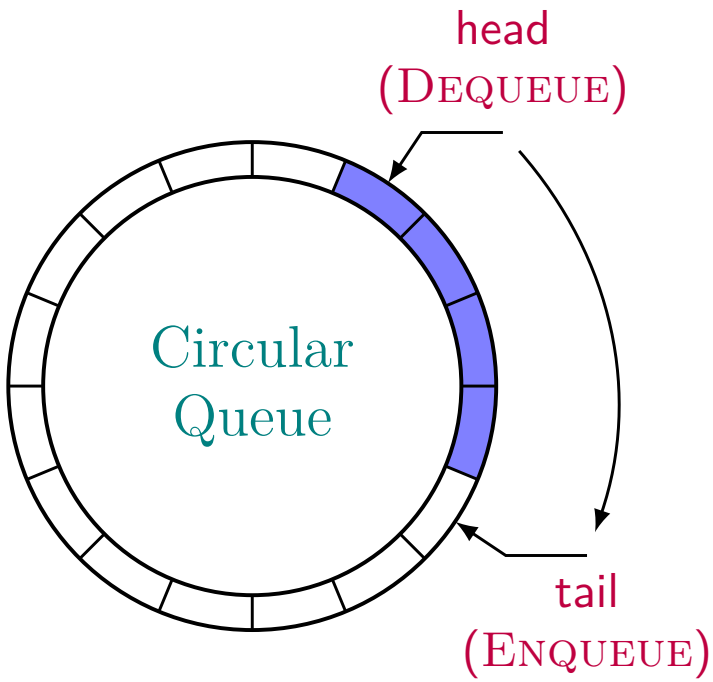


procedure DEQUEUE(Q)
  if $Q.head = Q.tail$ **then**
    return "UNDERFLOW"
  ...

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

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

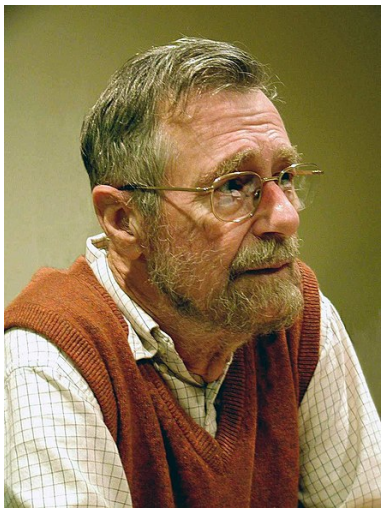
$[l, r)$ $(l, r]$ $[l, r]$ (l, r)

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) $2 \leqslant i < 13$
b) $1 < i \leqslant 12$
c) $2 \leqslant i \leqslant 12$
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**
            $Push\Big(S_2, Pop(S_1)\Big)$
    $Pop(S_2)$

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?

$$\textsc{Enq}(x, t_1), \textsc{Enq}(y, t_2) \wedge t_1 < t_2$$

$$\Longrightarrow$$

$$\textsc{Deq}(x, t_3), \textsc{Deq}(y, t_4) \wedge t_3 < t_4$$

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 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?

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

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

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

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

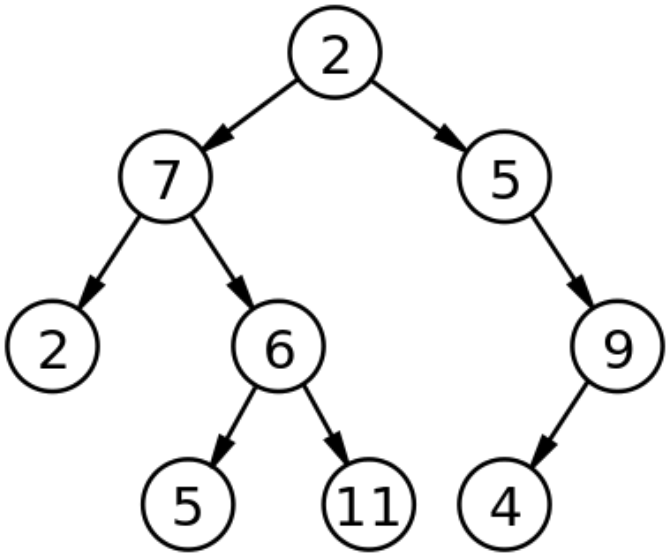
item	Push into S_1	Pop from S_1	Push into S_2	Pop from S_2
x	1	1	1	1

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

$$\hat{c}_{\text{ENQ}} = 3$$
$$\hat{c}_{\text{DEQ}} = 1$$

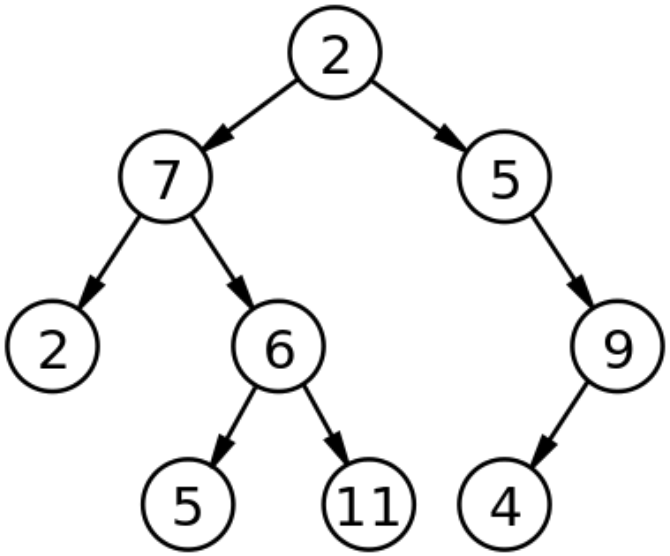
Recursive Binary Tree Traversal (Problem $10.4 - 2$)

$O(n)$



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

$O(n)$



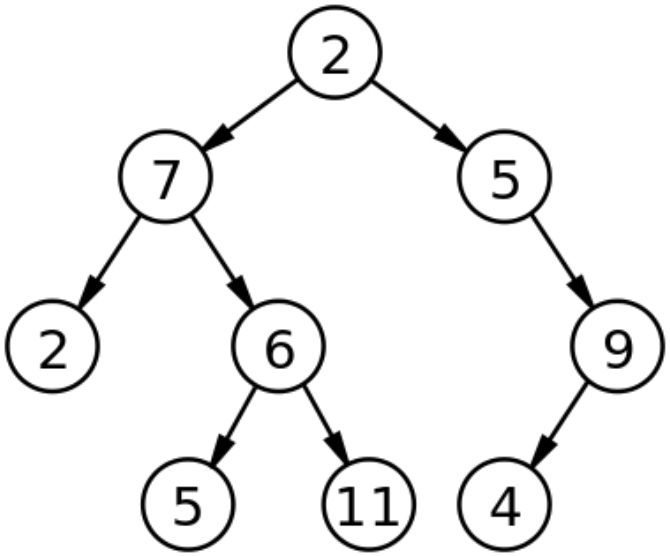
procedure RECURSIVE-DFS(t)
print $t.key$

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 $10.4 - 2$)

$O(n)$



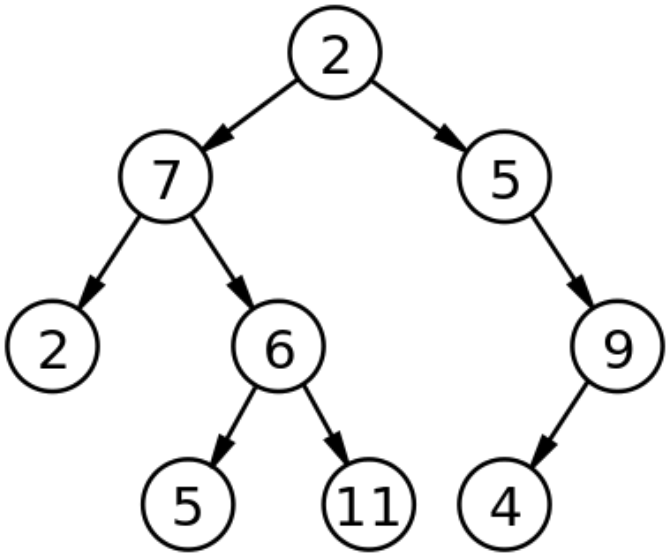
procedure RECURSIVE-DFS(t)
print $t.key$

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

RECURSIVE-DFS($T.root$)

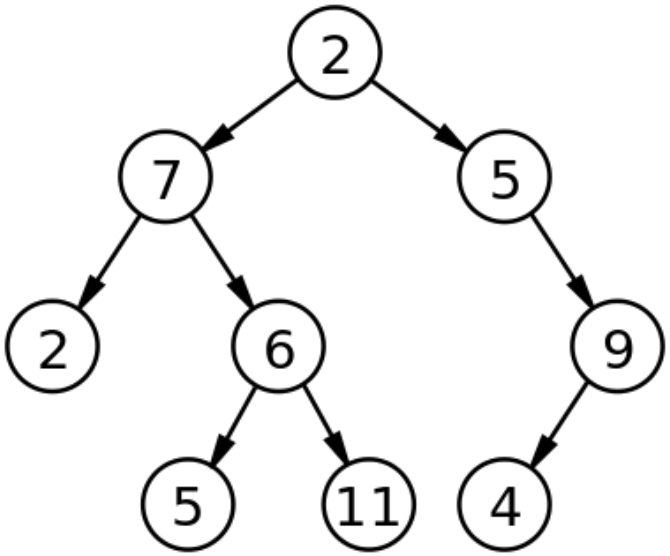
Non-recursive Binary Tree Traversal (Problem $10.4 - 2$)

$O(n)$



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

$O(n)$



procedure ITERATIVE-DFS(t)
  S.PUSH(t) ▷ S : stack

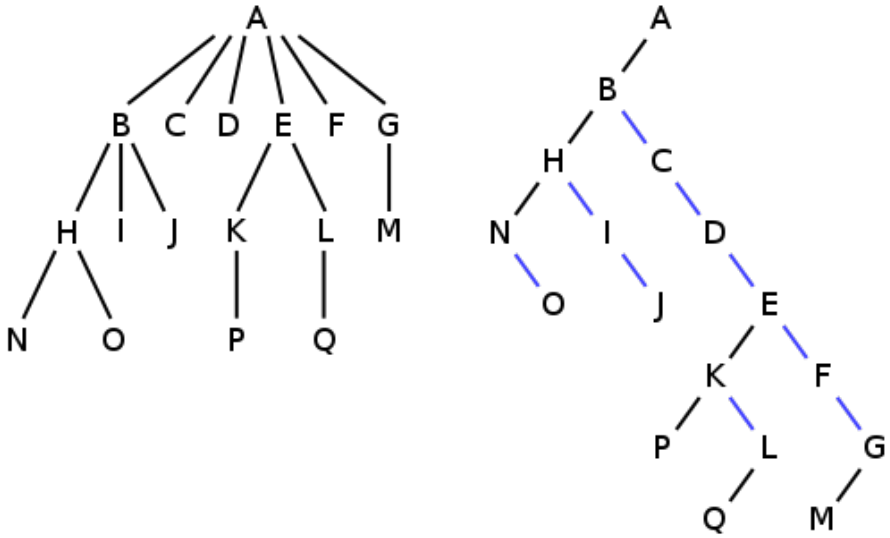
  while $S \neq \emptyset$ **do**
    $v \leftarrow S$.POP()
    print $v.key$

    if $v.right \neq$ NIL **then**
      S.PUSH($v.right$)
    if $v.left \neq$ NIL **then**
      S.PUSH($v.left$)

ITERATIVE-DFS($T.root$)

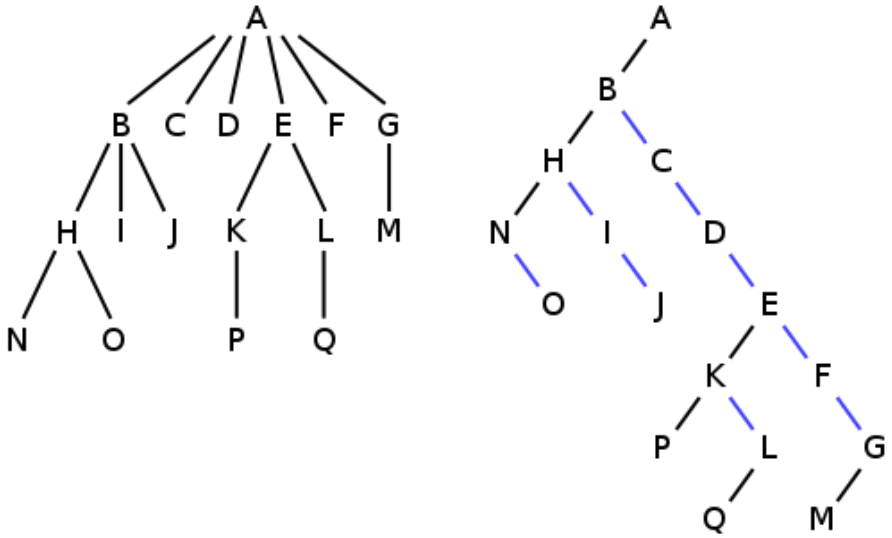
"LCRS" Tree Traversal (Problem $10.4 - 2$)

$$O(n)$$



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

$O(n)$

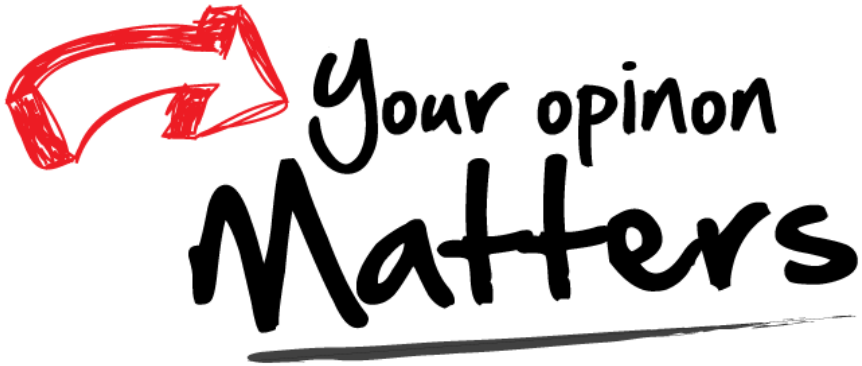


procedure RECURSIVE-DFS(t)
print $t.key$

if $t.lc \neq$ NIL **then**
RECURSIVE-DFS($t.lc$)
if $t.rs \neq$ NIL **then**
RECURSIVE-DFS($t.rs$)

RECURSIVE-DFS($T.root$)


Thank
You!



Office 302

Mailbox: H016

hfwei@nju.edu.cn