Data Structures and Algorithms

Fourth Practice

Stack representation

Let S be a stack containing numbers. We can implement a stack of at most n elements with an array S[1, . . . ,n]. The array has an attribute **S.top** that indexes the most recently inserted element.

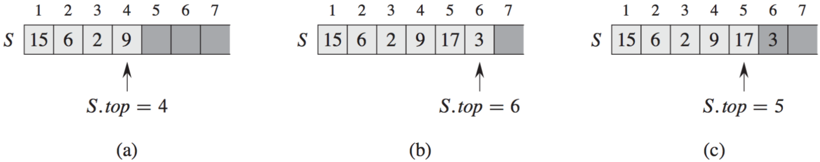


Figure 10.1 An array implementation of a stack S. Stack elements appear only in the lightly shaded positions. **(a)** Stack S has 4 elements. The top element is 9. **(b)** Stack S after the calls PUSH(S, 17) and PUSH(S, 3). **(c)** Stack S after the call POP(S) has returned the element 3, which is the one most recently pushed. Although element 3 still appears in the array, it is no longer in the stack; the top is element 17.

Stack representation – exercise 1

Using Figure 10.1 as a model, illustrate the result of each operation in the sequence

PUSH(S,3)
PUSH(S,9)
PUSH(S,2)
POP(S)
PUSH(S,1)
POP(S)
POP(S)
PUSH(S,4)
PUSH(S,7)

on an initially empty stack S stored in array S[1...6].

Stack representation – exercise 1, solution

Operation	1	2	3	4	5	6	S.top
PUSH(S,3)	3						1
PUSH(S,9)	3	9					2
PUSH(S,2)	3	9	2				3
POP(S)	3	9	✓				2
PUSH(S,1)	3	9	1				3
POP(S)	3	9					2
POP(S)	3						1
PUSH(S,4)	3	4					2
PUSH(S,7)	3	4	7				3

Stack representation – exercise 2

Using Figure 10.1 as a model, illustrate the result of each operation in the sequence

PUSH(S,1)
PUSH(S,5)
POP(S)
PUSH(S,2)
POP(S)
POP(S)
PUSH(S,3)
POP(S)
POP(S)

on an initially empty stack S stored in array S[1...5].

Stack representation – exercise 2, solution

Operation	1	2	3	4	5	S.top
PUSH(S,1)	1					1
PUSH(S,5)	1	5				2
POP(S)	1					1
PUSH(S,2)	1	2				2
POP(S)	1					1
POP(S)						0
PUSH(S,3)	3					1
POP(S)						0
POP(S)						underflow

Stack representation – exercise 3

Using Figure 10.1 as a model, illustrate the result of each operation in the sequence

PUSH(S,8)
PUSH(S,6)
POP(S)
PUSH(S,5)
PUSH(S,1)
PUSH(S,9)
POP(S)
PUSH(S,2)
PUSH(S,4)
PUSH(S,7)

on an initially empty stack S stored in array S[1...5].

Stack representation – exercise 3, solution

Operation	1	2	3	4	5	S.top
PUSH(S,8)	8					1
PUSH(S,6)	8	6				2
POP(S)	8					1
PUSH(S,5)	8	5				2
PUSH(S,1)	8	5	1			3
PUSH(S,9)	8	5	1	9		4
POP(S)	8	5	1			3
PUSH(S,2)	8	5	1	2		4
PUSH(S,4)	8	5	1	2	4	5
PUSH(S,7)	8	5	1	2	4	overflow

Stack representation – exercise 4

Implement two stacks in one array A[1...n] in such a way that neither stack overflows unless the total number of elements in both stacks together is n. The PUSH and POP operations should run in $\mathcal{O}(1)$ time.

Stack representation – exercise 4, solution

We will call the stacks T and R. Initially, set $T.top = 0$ and $R.top = n + 1$. Essentially, stack T uses the first part of the array and stack R uses the last part of the array. In stack T, the top is the rightmost element of T. In stack R, the top is the leftmost element of R.

Algorithm 1 PUSH(S,x)

if $S == T$ **then**
    if $T.top + 1 == R.top$ **then**
        error “overflow”
    else
        $T.top = T.top + 1$
        $T[T.top] = x$
    end if
end if
if $S == R$ **then**
    if $R.top - 1 == T.top$ **then**
        error “overflow”
    else
        $R.top = R.top - 1$
        $T[T.top] = x$
    end if
end if

Stack representation – exercise 4, solution

Algorithm 2 POP(S)

```
if S == T then
    if T.top == 0 then
        error "underflow"
    else
        T.top = T.top − 1.
        return T[T.top + 1]
    end if
end if
if S == R then
    if R.top == n + 1 then
        error "underflow"
    else
        R.top = R.top + 1.
        return R[R.top − 1]
    end if
end if
```

Queue representation

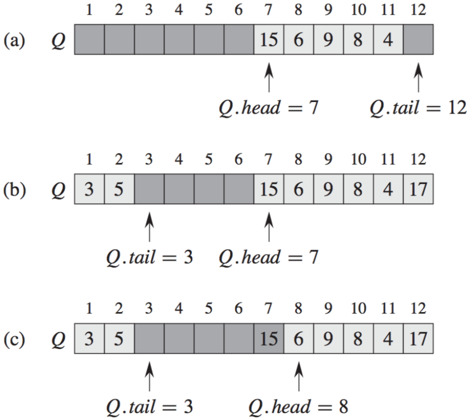


Figure 10.2 A queue implemented using an array $Q[1..12]$. Queue elements appear only in the lightly shaded positions. **(a)** The queue has 5 elements, in locations $Q[7..11]$. **(b)** The configuration of the queue after the calls ENQUEUE(Q, 17), ENQUEUE(Q, 3), and ENQUEUE(Q, 5). **(c)** The configuration of the queue after the call DEQUEUE(Q) returns the key value 15 formerly at the head of the queue. The new head has key 6.

Queue representation – exercise 1

Using Figure 10.2 as a model, illustrate the result of each operation in the sequence

ENQUEUE(Q,4)
DEQUEUE(Q)
ENQUEUE(Q,7)
ENQUEUE(Q,2)
ENQUEUE(Q,9)
ENQUEUE(Q,5)
ENQUEUE(Q,3)
DEQUEUE(Q)
ENQUEUE(Q,1)

on an initially empty cyclic queue Q stored in array Q[1...6].

Queue representation – exercise 1, solution

Operation	1	2	3	4	5	6	Q.head	Q.tail
ENQUEUE(Q,4)	4						1	2
DEQUEUE(Q)							2	2
ENQUEUE(Q,7)		7					2	3
ENQUEUE(Q,2)		7	2				2	4
ENQUEUE(Q,9)		7	2	9			2	5
ENQUEUE(Q,5)		7	2	9	5		2	6
ENQUEUE(Q,3)		7	2	9	5	3	2	1
DEQUEUE(Q)			2	9	5	3	3	1
ENQUEUE(Q,1)	1		2	9	5	3	3	2

Queue representation – exercise 2

Using Figure 10.2 as a model, illustrate the result of each operation in the sequence

ENQUEUE(Q,5)
ENQUEUE(Q,1)
ENQUEUE(Q,7)
DEQUEUE(Q)
ENQUEUE(Q,3)
ENQUEUE(Q,8)
DEQUEUE(Q)
ENQUEUE(Q,2)
ENQUEUE(Q,4)

on an initially empty cyclic queue Q stored in array Q[1...5].

Queue representation – exercise 2, solution

Operation	1	2	3	4	5	Q.head	Q.tail
ENQUEUE(Q,5)	5					1	2
ENQUEUE(Q,1)	5	1				1	3
ENQUEUE(Q,7)	5	1	7			1	4
DEQUEUE(Q)		1	7			2	4
ENQUEUE(Q,3)		1	7	3		2	5
ENQUEUE(Q,8)		1	7	3	8	2	1
DEQUEUE(Q)			7	3	8	3	1
ENQUEUE(Q,2)	2		7	3	8	3	2
ENQUEUE(Q,4)	2	✓	7	3	8	overlow	

Queue representation – exercise 3

Using Figure 10.2 as a model, illustrate the result of each operation in the sequence

ENQUEUE(Q,a)
ENQUEUE(Q,c)
DEQUEUE(Q)
ENQUEUE(Q,b)
DEQUEUE(Q)
DEQUEUE(Q)
ENQUEUE(Q,d)
DEQUEUE(Q)
DEQUEUE(Q)

on an initially empty cyclic queue Q stored in array Q[1...7].

Queue representation – exercise 3, solution

Operation	1	2	3	4	5	6	7	Q.head	Q.tail
ENQUEUE(Q,a)	a							1	2
ENQUEUE(Q,c)	a	c						1	3
DEQUEUE(Q)		c						2	3
ENQUEUE(Q,b)		c	b					2	4
DEQUEUE(Q)			b					3	4
DEQUEUE(Q)								4	4
ENQUEUE(Q,d)				d				4	5
DEQUEUE(Q)								5	5
DEQUEUE(Q)								underflow	

Queue representation – exercise 4

Rewrite ENQUEUE and DEQUEUE to detect underflow and overflow of a queue.

Queue representation – exercise 4, solution

Algorithm 3 ENQUEUE

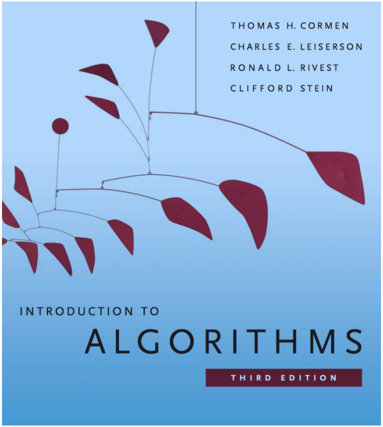
if $Q.head == Q.tail + 1$, or $Q.head == 1$ and $Q.tail == Q.length$ **then**
    error "overflow"
end if
$Q[Q.tail] = x$
if $Q.tail == Q.length$ **then**
    $Q.tail = 1$
else
    $Q.tail = Q.head + 1$
end if

Queue representation – exercise 4, solution

Algorithm 4 DEQUEUE

if $Q.tail == Q.head$ **then**
  error "underflow"
end if
$x = Q[Q.head]$
if $Q.head == Q.length$ **then**
  $Q.head = 1$
else
  $Q.head = Q.head + 1$
end if
return x

References


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION