Advanced Algorithms and Data Structures

Exercises: Stacks and Queues

Exercise 1

- Illustrate the result of each operation in the sequence PUSH(S, 4), PUSH(S, 1), PUSH(S, 3), POP(S), PUSH(S, 8), and POP(S) on an initially empty stack S stored in array S[1..6].

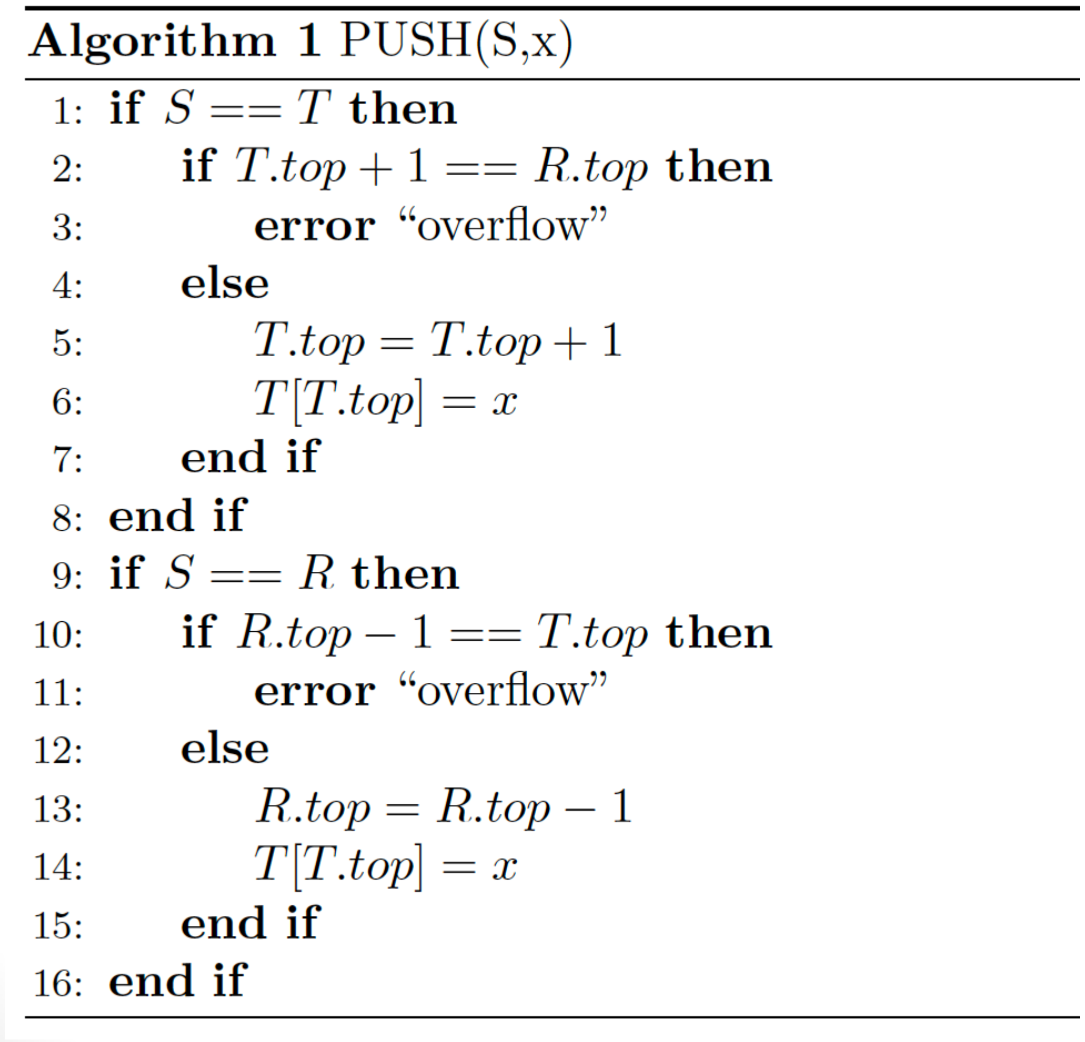
Answer

- 4
- 4 → 1
- 4 → 1 → 3
- 4 → 1
- 4 → 1 → 8
- 4 → 1

Exercise 2

- Explain how to 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 O(1) time.

Answer



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
```

Answer

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
```

Exercise 3

- Illustrate the result of each operation in the sequence ENQUEUE(Q, 4), ENQUEUE(Q, 1), ENQUEUE(Q, 3), DEQUEUE(Q), ENQUEUE(Q, 8), and DEQUEUE(Q) on an initially empty queue Q stored in array Q[1..6].

Answer

- 4
- 4 → 1
- 4 → 1 → 3
- 1 → 3
- 1 → 3 → 8
- 3 → 8

Exercise 4

- Whereas a stack allows insertion and deletion of elements at only one end, and a queue allows insertion at one end and deletion at the other end, a *deque* (double-ended queue) allows insertion and deletion at both ends.
- Write four O(1)-time procedures to insert elements into and delete elements from both ends of a deque implemented by an array.

Answer

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 $x$
```

Answer

Algorithm 5 HEAD-ENQUEUE(Q,x)

```
$Q[Q.head] = x$
**if** $Q.head == 1$ **then**
    $Q.head = Q.length$
**else**
    $Q.head = Q.head - 1$
**end if**
```

Algorithm 6 TAIL-ENQUEUE(Q,x)

```
$Q[Q.tail] = x$
**if** $Q.tail == Q.length$ **then**
    $Q.tail = 1$
**else**
    $Q.tail = Q.tail + 1$
**end if**
```

Answer

Algorithm 7 HEAD-DEQUEUE(Q,x)

```
$x = Q[Q.head]$
if $Q.head == Q.length$ then
    $Q.head = 1$
else
    $Q.head = Q.head + 1$
end if
```

Algorithm 8 TAIL-DEQUEUE(Q,x)

```
$x = Q[Q.tail]$
if $Q.tail == 1$ then
    $Q.tail = Q.length$
else
    $Q.tail = Q.tail - 1$
end if
```

Exercise 5

- Show how to implement a queue using two stacks. Analyze the running time of the queue operations.

Answer

- The operation enqueue will be the same as pushing an element on to stack 1. This operation is O(1). To dequeue, we pop an element from stack 2. If stack 2 is empty, for each element in stack 1 we pop it off, then push it on to stack 2. Finally, pop the top item from stack 2. This operation is O(n) in the worst case.

Exercise 6

- Show how to implement a stack using two queues. Analyze the running time of the stack operations.

Answer

- The following is a way of implementing a stack using two queues, where pop takes linear time, and push takes constant time. The first of these ways, consists of just enqueueing each element as you push it. Then, to do a pop, you dequeue each element from one of the queues and place it in the other, but stopping just before the last element. Then, return the single element left in the original queue.