Data Structures and Algorithms in Java™

Sixth Edition

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Instructor's Solutions Manual

WILEY

Chapter

6 Stacks, Queues, and Deques

Hints and Solutions

Reinforcement

R-6.1) Hint If a stack is empty when pop is called, its size does not change.

R-6.1) Solution The size of the stack is $25-10+3=18$.

R-6.2) Hint It is one less than the size of S.

R-6.2) Solution 17.

R-6.3) Hint Use a paper and pencil with eraser to simulate the stack.

R-6.3) Solution 3, 8, 2, 1, 6, 7, 4, 9

R-6.4) Hint Transfer items one at a time.

R-6.4) Solution

```
static <E> void transfer(Stack<E> S, Stack<E> T) {
  while (!S.isEmpty()) {
    T.push(S.pop());
  }
}
```

R-6.5) Hint First check if the stack is already empty.

R-6.5) Solution If the stack is empty, then return (the stack is empty). Otherwise, pop the top element from the stack and recur.

R-6.6) Hint Give a recursive definition.

R-6.6) Solution An arithmetic expression has matching grouping symbols if it has one of the following structures, where S denotes an arithmetic expression without grouping symbols, and E, E', and E'' recursively denote an arithmetic expression with matching grouping symbols:

- S
- $E'(E)E''$
- $E'[E]E''$
- $E'\{E\}E''$

R-6.7) Hint If a queue is empty when dequeue is called, its size does not change.

R-6.7) Solution The size of the queue is $32 - 15 + 5 = 22$.

R-6.8) Hint Each successful dequeue operation causes that index to shift circularly to the right.

R-6.8) Solution f=10.

R-6.9) Hint Use a paper and pencil with eraser to simulate the queue.

R-6.9) Solution 5, 3, 2, 8, 9, 1, 7, 6

R-6.10) Hint Use operations at the appropriate ends of the deque.

R-6.11) Hint Use operations at the appropriate ends of the deque.

R-6.12) Hint Use a paper and pencil to simulate the deque.

R-6.12) Solution 9, false, 9, 2, 7, 6, 2, 1

R-6.13) Hint Use the results of removal methods as arguments to insertion methods.

R-6.13) Solution

D.addLast(D.removeFirst())
D.addLast(D.removeFirst())
D.addLast(D.removeFirst())
Q.enqueue(D.removeFirst())
Q.enqueue(D.removeFirst())
D.addFirst(Q.dequeue())
D.addFirst(Q.dequeue())
D.addFirst(D.removeLast())
D.addFirst(D.removeLast())
D.addFirst(D.removeLast())

R-6.14) Hint Use the results of removal methods as arguments to insertion methods. In addition, you will need to use more of the stack for temporary storage.

R-6.14) Solution

D.addLast(D.removeFirst())
D.addLast(D.removeFirst())
D.addLast(D.removeFirst())
S.push(D.removeFirst())
D.addLast(D.removeFirst())
D.addFirst(Spop())
D.addFirst(D.removeLast())
D.addFirst(D.removeLast())
D.addFirst(D.removeLast())
D.addFirst(D.removeLast())

R-6.15) Hint You might start by concatenating the bodies of the dequeue and enqueue methods and then look to avoid redundancy.

R-6.15) Solution

```
public void rotate( ) {
  if (sz < data.length) {
    // object must move within the array
    int avail = (f + sz) % data.length;
    data[avail] = data[f];
    data[f] = null;
  }
  f = (f + 1) % data.length;
}
```

Creativity

C-6.16) Hint Pop the top integer, but remember it.

C-6.16) Solution

```
x = S.pop( );
if (x < S.top( ))
  x = S.pop( );
```

Note that if the largest integer is the first or second element of S, then x will store it. Thus, x stores the largest element with probability $2/3$.

C-6.17) Hint You will need to do three transfers.

C-6.17) Solution

```
transfer(S, A);    // contents in A are in reverse order as original
transfer(A, B);    // contents in B are in same order as original
transfer(B, S);    // contents in S are in reverse order as original
```

C-6.18) Hint After finding what's between the $<$ and $>$ characters, the tag is only the part before the first space (if any).

C-6.19) Hint Use a stack.

C-6.20) Hint You can still use R as temporary storage, as long as you never pop its original contents.

C-6.20) Solution Let r, s and t denote the original sizes of the stacks.

Make s calls to R.push(S.pop( ))
Make t calls to R.push(T.pop( ))
Make $s+t$ calls to S.push(R.pop( ))

C-6.21) Hint Use a stack to reduce the problem to that of enumerating all permutations of the numbers $\{1,2,\ldots,n-1\}$.

C-6.22) Hint Think of the stacks like jugs and the dump operations like water being poured between two jugs.

C-6.22) Solution

```
dump(A,B)            // A=95, B=5, C=0
dump(B,C)            // A=95, B=2, C=3
dump(C,A)            // A=98, B=2, C=0
dump(B,C)            // A=98, B=0, C=2
dump(A,B)            // A=93, B=5, C=2
dump(B,C)            // A=93, B=4, C=3
```

C-6.23) Hint Use the stack to store the elements yet to be used to generate subsets and use the queue to store the subsets generated so far.

C-6.24) Hint Think of how you might use Q to process the elements of S twice.

C-6.24) Solution The solution is to actually use the queue Q to process the elements in two phases. In the first phase, we iteratively pop each the element from S and enqueue it in Q, and then we iteratively dequeue each element from Q and push it into S. This reverses the elements in S. Then we repeat this same process, but this time we also look for the element x. By passing the elements through Q and back to S a second time, we reverse the reversal, thereby putting the elements back into S in their original order.

C-6.25) Hint Rotate elements within the queue.

C-6.25) Solution One approach to implement the stack ADT using a queue Q, simply enqueues elements into Q whenever a push call is made. This takes $O(1)$ time to complete. For pop calls, we make $n-1$ calls to $\mathsf{Q.enqueue(Q.dequeue())}$, where n is the current size, and then return $\mathsf{Q.dequeue()}$, as that is the most recently inserted element. This requires $O(n)$ time. We can use a similar approach for top, but rotating the answer back to the end of the queue.

A better approach is to align the elements of the queue so that the "top" is aligned with the front of the queue. To push an element, we enqueue it and then immediately make $n-1$ calls to $\mathsf{Q.enqueue(Q.dequeue())}$. This requires $O(n)$ time, but with that orientation, both pop and top can be implemented in $O(1)$ time by a respective call to dequeue or front.

C-6.26) Hint You can try it out

C-6.26) Solution The variable f could have been initialized to any valid index from 0 to $\mathsf{capacity} - 1$, since the front of the queue need not be initially aligned with the front of the array. Of course, since different queues will have different capacities, 0 is a convenient choice for all.

C-6.27) Hint See Section 3.6 for a discussion of cloning data structures.

C-6.27) Solution

```
public ArrayStack<E> clone() throws CloneNotSupportedException {
  ArrayStack<E> other = (ArrayStack<E>) super.clone();
  other.data = data.clone();
  return other;
}
```

C-6.28) Hint See Section 3.6 for a discussion of cloning data structures.

C-6.28) Solution Similarly to previous solution for ArrayStack.

C-6.29) Hint You are welcome to modify the SinglyLinkedList class to add necessary support

C-6.29) Solution Such a method could be implemented by calling list.concatenate(Q2.list) if we add the following method to the SinglyLinkedList class:

```
/**
 * Append all elements of other to the end of this list.
 * The other list will become empty as a result.
 */
public void concatenate(SinglyLinkedList<E> other) {
  if (head == null) {
    head = other.head;
  } else {
    tail.setNext(other.head);
  }
  tail = other.tail;
  size += other.size;
  // clear other list
  other.head = other.tail = null;
  other.size = 0;
}
```

C-6.30) Hint Use separate indices for the two ends.

C-6.31) Hint Think of using one stack for each end of the deque.

C-6.32) Hint Use the deque like a stack.

C-6.33) Hint Think of the queues like boxes and the integers like red and blue marbles.

C-6.33) Solution Alice should put on even integer in C and all the other 99 integers in D. This gives her a $74/99$ (roughly 74.7%) chance of winning.

C-6.34) Hint Lazy and Crazy should only go across once.

C-6.34) Solution Mazie and Daisy go across (4 min.), Daisy comes back (4 min.), Crazy and Lazy go across (20 min.), Mazie comes back (2 min.), and then Mazie and Daisy go across (4 min.).

Projects

P-6.35) Hint You will need to use a stack.

P-6.36) Hint Define two stacks that are used match sell orders with buy orders.

P-6.37) Hint Start one stack at each end of the array, growing toward the center.

P-6.38) Hint How does this functionality compare to a deque?

P-6.39) Hint Think carefully about the orientation of the linked list.

P-6.40) Hint The section on the Deque ADT gives advice on using a circular array implementation.