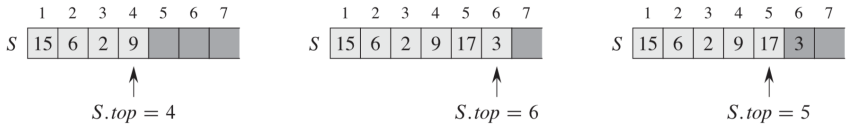
Intro Algo Chapter 10

Mengxi Wu(mw4355@nyu.edu)

September 27, 2020

1 Stack

Stack is a Last in First out data structure. It has method Push (insert), Delete. It also has attribute S.top that indexes the most recently inserted element. When S.top = 0, the stack is empty. If we use pop on an empty stack, we say stack underflows. If S.top exceed the dimension of the array, the stack overflows.



2 Queue

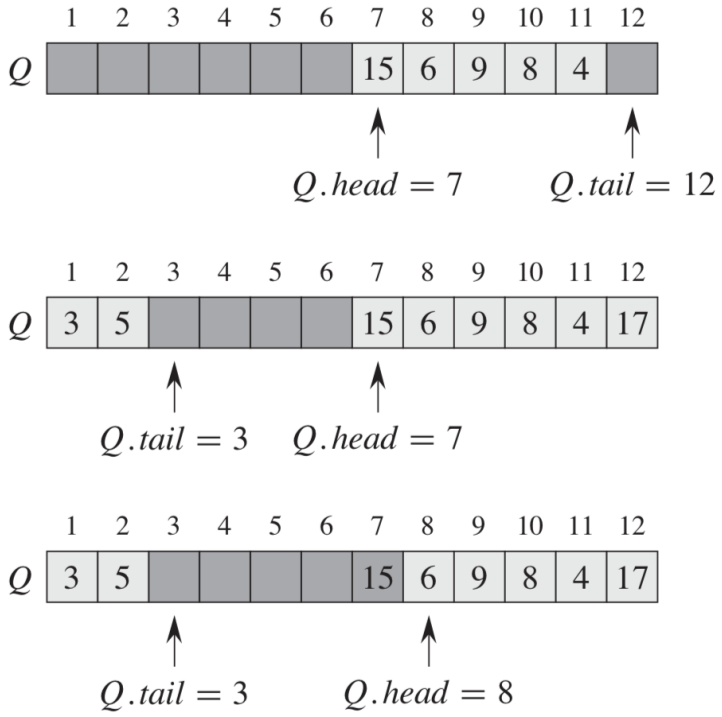
Queue is a First in Frist out data structure. It has insert operation which calls enqueue and the delete operation which calls dequeue. The queue has a head and a tail. The elements in the queue reside in locations Q.head, Q.head+1,...Q.tail-1. The queue is empty when Q.head == Q.tail. The queue is full when Q.head == 1 and Q.tail == Q.length, or Q.head = Q.tail + 1.

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
```

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
```



3 Linked Lists

The single linked list has two objects in one node the value and the next pointer. The double linked list has three objects in one node the value, the next pointer, and the prev pointer. In the circular linked list, the prev pointer of the head of the list points to the tail, and the next pointer of the tail of the list points to the head.

Method 1: Search through a linked list. In worst case, it takes $\Theta(n)$, since it may iterates through the entire list.

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3       x = x.next
4  return x
```

Method 2: Insert an element in a double linked list. The running time for list insert is $\Theta(1)$.

```
LIST-INSERT(L, x)
1  x.next = L.head
2  if L.head ≠ NIL
3       L.head.prev = x
4  L.head = x
5  x.prev = NIL
```

Method 3: Delete an element in a linked list. It requires we to obtain the pointer that points to the element. So, we need to fist apply search method to find the pointer that points to the target value.The operation to delete the element takes $\Theta(1)$ if the pointer is given, but considering search method, it requires $\Theta(n)$.

LIST-DELETE(L, x)

```
1  if x.prev ≠ NIL
2        x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5        x.next.prev = x.prev
```

Another way to handle linked list is with sentinels. For example, represent Nil in a list as L.nil (a node with value as Nil). L:nil lies between the head and tail. L.nil.next points to the head of the list, and L.nil.prev points to the tail. The next attribute of the tail and the prev the head point to L.nil. Sentinels rarely reduce the asymptotic time bounds of data structure operations, but they can reduce constant factors. We save only $O(1)$ time in the LIST-INSERT and LIST-DELETE procedures. In the following figure, L.nil is the first dark node in the list (the one without number and L.nil is pointed out the arrow).