Queues and Singly Linked Lists

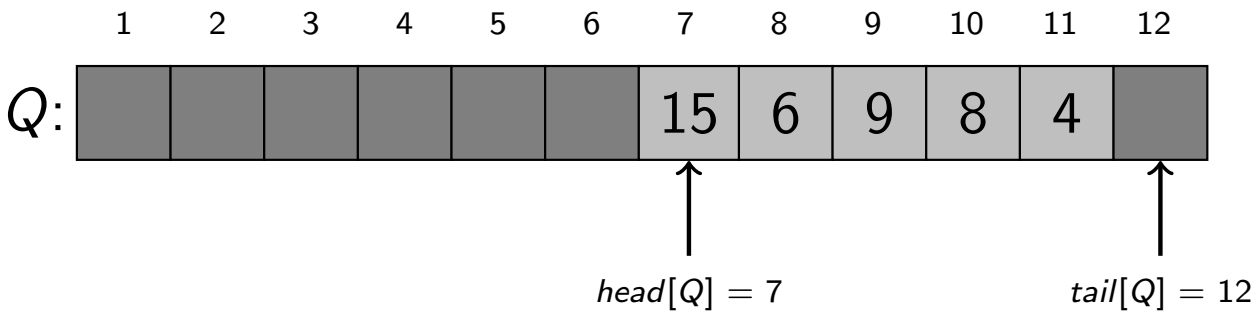
Subhabrata Samajder

IIIT, Delhi

Summer Semester,

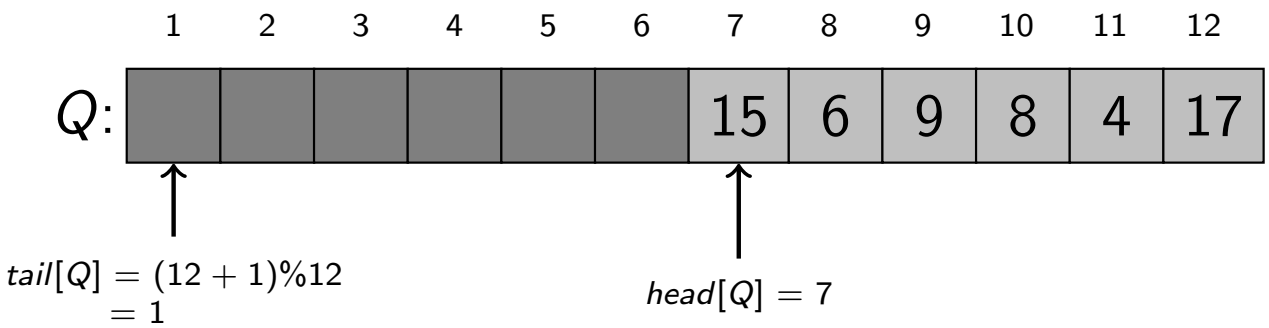
23rd May, 2022

An Improvement



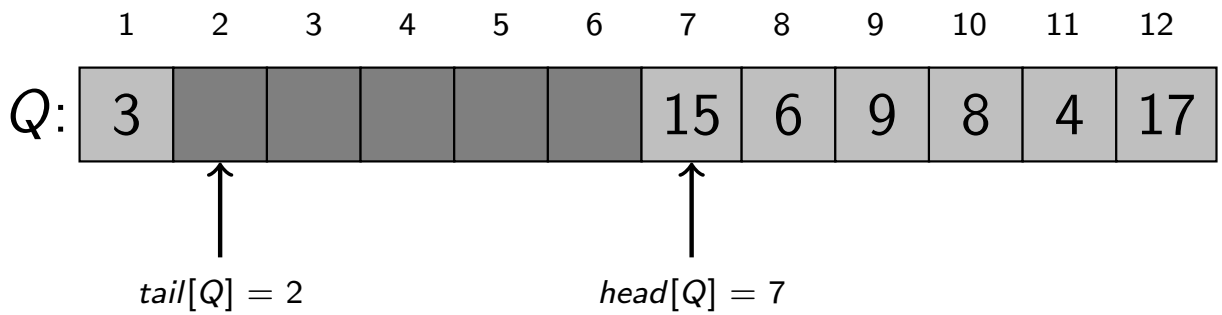
An Improvement

ENQUEUE$(Q, 17)$:



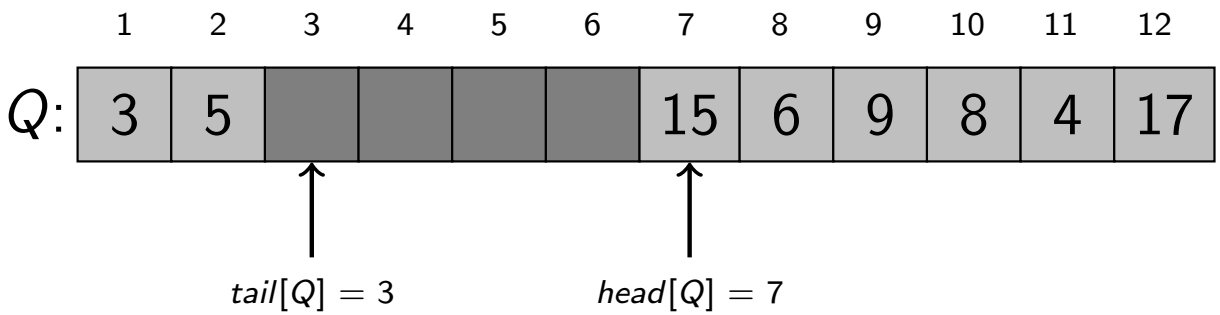
An Improvement

ENQUEUE$(Q, 3)$:



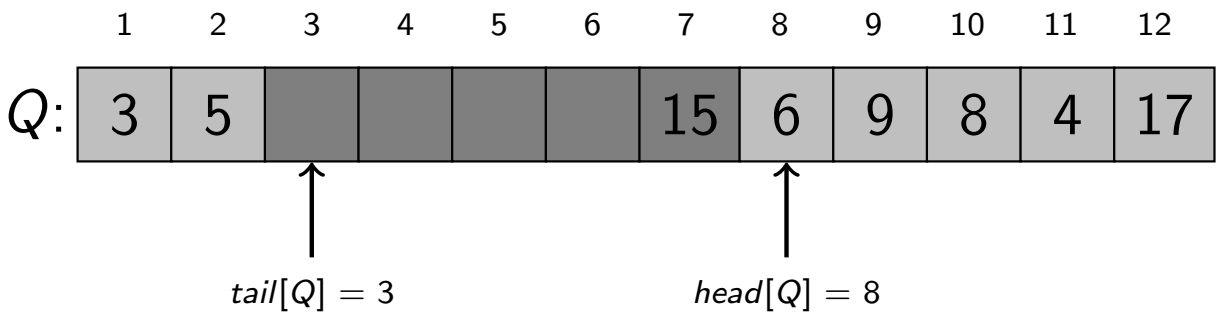
An Improvement

ENQUEUE$(Q, 5)$:



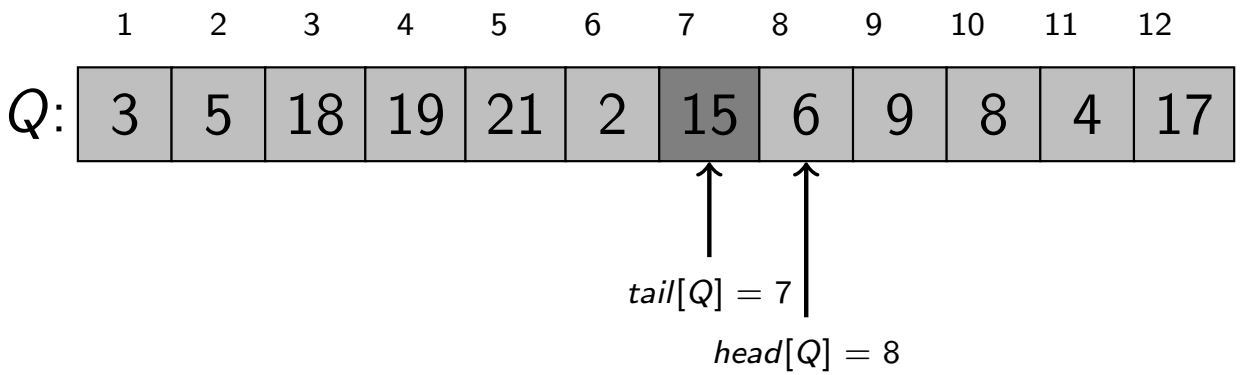
An Improvement

DEQUEUE(Q):



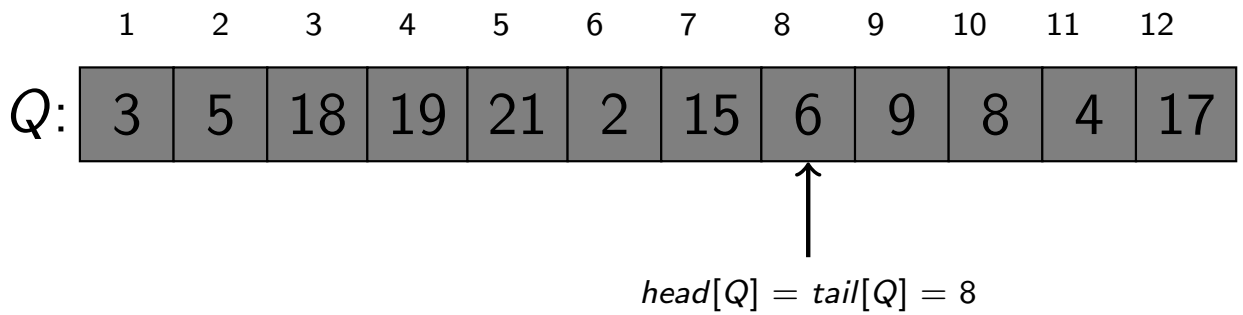
An Improvement

Queue Full: $head[Q] = (tail[Q] + 1) \mod n$.
Error: Overflow.



An Improvement

Queue Empty: $head[Q] = tail[Q]$.
Error: Underflow.



ENQUEUE(Q, x)

Begin
  If $(head[Q] = (tail[Q] + 1) \mod n)$
    return "**overflow error**"

  $Q[tail[Q]] \leftarrow x$;

  If $(tail[Q] = length[Q])$
    $tail[Q] \leftarrow 1$;
  Else
    $tail[Q] \leftarrow tail[Q] + 1$;

End

DEQUEUE(Q)

```
Begin
  If (head[Q] = tail[Q])
    return "underflow error"

  x ← Q[head[Q]];

  If (tail[Q] = length[Q])
    head[Q] ← 1;
  Else
    head[Q] ← head[Q] + 1;

  return x;
End
```

Applications of Queues

- Access to shared resources (e.g., printer).
- Simulations of read world situations of waiting lines (bank teller, flight bookings).
- To efficiently maintain a First-in-first out (FIFO) order on some entities
- In a multitasking operating system, the CPU cannot run all jobs at once, so jobs must be batched up and then scheduled according to order in a queue.
- User input in a game

A C Implemention of a Queue Using An Array

Initialization

```
/* Queue */
int main() {
   int head, tail;
   int Q[len];

   /* Initialisation */
   head = tail = 0;
   .
   .
   .
}
```

Enqueue

Insert an element at the tail of the queue Q and redefine tail:

```
/* Enqueue */
int Enqueue(int data, int *Q) {
   /* check if queue is full or not */
   if (head == (tail + 1)% length) {
      printf("\n ERROR: Queue is full\n");
      return FLAG;
   }
   /* insert element at the tail */
   else {
      Q[tail] = data;
      tail = (tail + 1)% length;
   }
   return 0;
}
```

Dequeue

Delete and return the element pointed by head of the queue:

```
/* Dequeue */
int Dequeue(int *Q) {
   int x;

   if (head == tail) {    // if queue is empty
      printf("\n ERROR: Queue is empty\n");
      return FLAG;
      }
   /* delete element from the head */
   else {
      x = Q[head];
      head = (head + 1)% length;
   }
   return x;
}
```

Front

Return the front element from the queue (if queue is not empty) but do not remove it.

```
/* prints the head of the queue */
void Front() {
   if (head == tail) {
      printf("\n Q is Empty\n");
      return FLAG;
   }

   printf("\n Front Element is: %d", Q[head]);
   return 0;
}
```

Exercise

Describe the output and final structure of the queue after the following operations:

- ENQUEUE(8)
- ENQUEUE(3)
- DEQUEUE()
- ENQUEUE(2)
- ENQUEUE(5)
- DEQUEUE()
- DEQUEUE()
- ENQUEUE(9)
- ENQUEUE(1)