Algoritmi e Strutture di Dati I

Massimo Franceschet

http://www.sci.unich.it/~francesc

Pila (stack)

La struttura di dati **pila** rappresenta un insieme dinamico nel quale le operazioni di inserimento e cancellazione seguono una politica

LIFO = Last In First Out

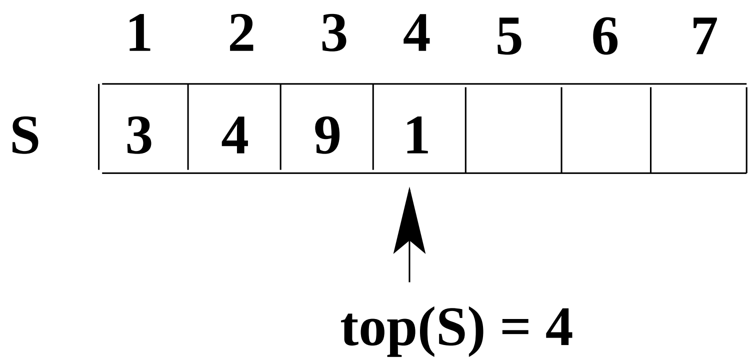
Ciò significa che l'oggetto cancellato è quello che è stato inserito **per ultimo**.

Pila (stack)

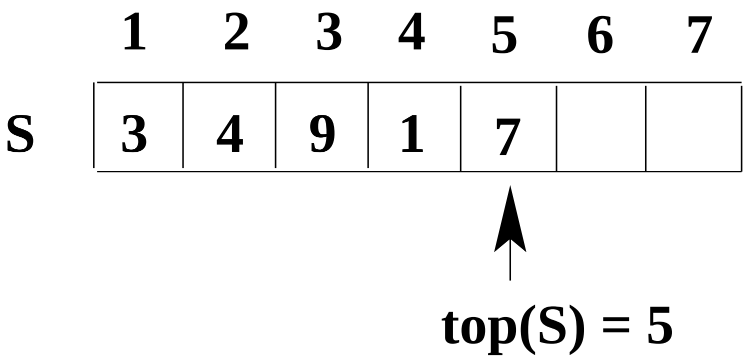
Una pila S possiede un attributo $top(S)$ che punta alla cima della pila e le seguenti quattro operazioni:

- $StackEmpty(S)$ che controlla se la pila S è vuota;
- $StackFull(S)$ che controlla se la pila S è piena;
- $Push(S, x)$ che inserisce l'oggetto x in cima alla pila S;
- $Pop(S)$ che rimuove l'oggetto che si trova in cima alla pila S;

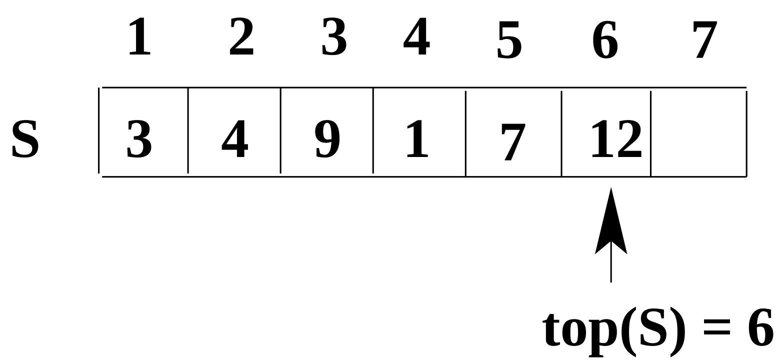
Una pila S



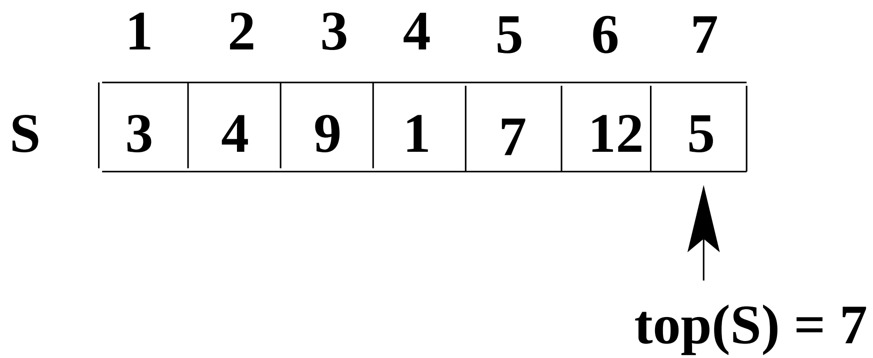
$\boxed{Push(S, 7)}$



$Push(S, 12)$



$\boxed{Push(S, 5)}$

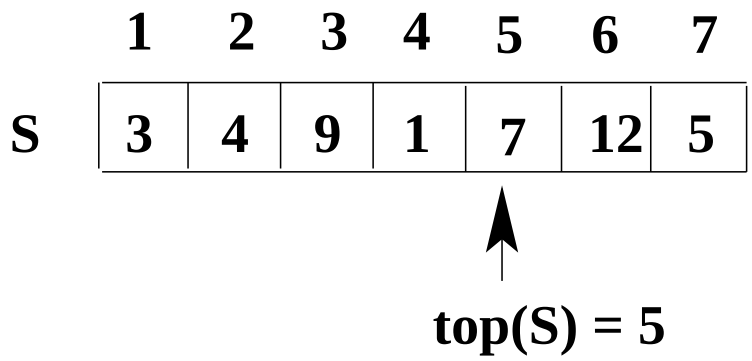


$Pop(S)$

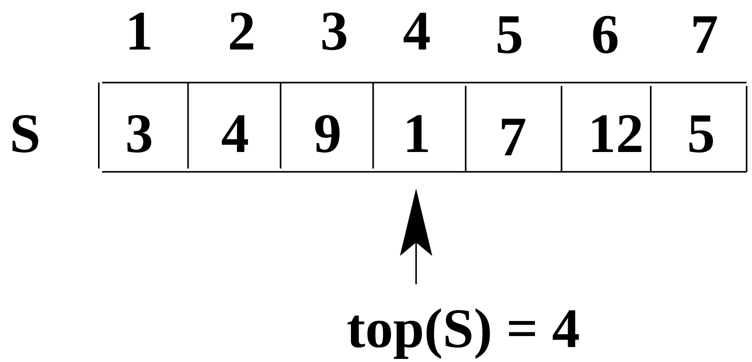
	1	2	3	4	5	6	7
S	**3**	**4**	**9**	**1**	**7**	**12**	**5**

top(S) = 6

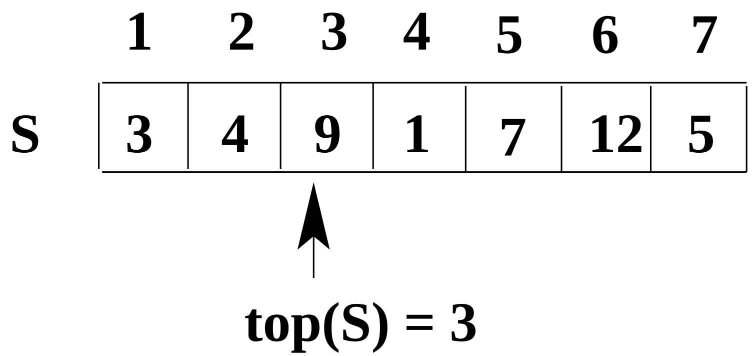
$Pop(S)$



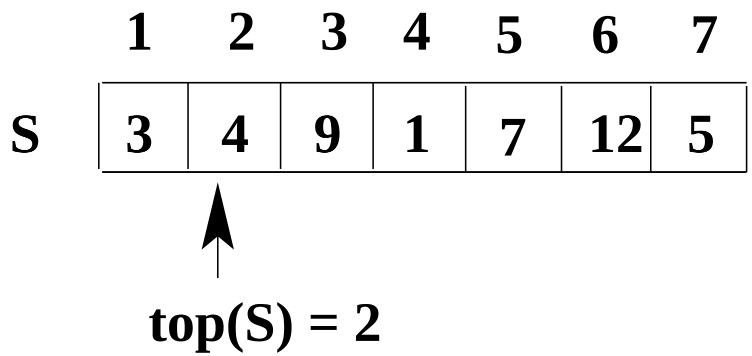
$Pop(S)$



$Pop(S)$



$Pop(S)$



$Pop(S)$

	1	2	3	4	5	6	7
S	**3**	**4**	**9**	**1**	**7**	**12**	**5**

top(S) = 1

$\boxed{Pop(S)}$

	1	2	3	4	5	6	7
S	**3**	**4**	**9**	**1**	**7**	**12**	**5**

top(S) = 0

StackEmpty(S)

if $top(S) = 0$ **then**
    return TRUE
else
    return FALSE
end if

StackFull(S)

if $top(S) = length(S)$ **then**
    return TRUE
else
    return FALSE
end if

Push(S,x)

if $StackFull(S)$ **then**
  error OVERFLOW
else
  $top(S) \leftarrow top(S) + 1$
  $S[top(S)] \leftarrow x$
end if

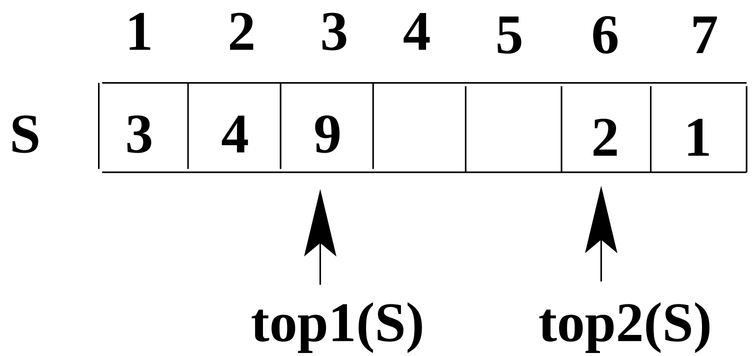
Pop(S)

if $StackEmpty(S)$ **then**
  error UNDERFLOW
else
  $x \leftarrow S[top[S]]$
  $top(S) \leftarrow top(S) - 1$
  return x
end if

Cosa fa?

while **not** $StackEmpty(R)$ **do**
  $Push(S, Pop(R))$
end while

Esercizio *Utilizzare un vettore di n elementi per rappresentare due pile in maniera che nessuna delle due pile vada in overflow se il numero totale degli elementi delle due pile è inferiore a n.*



StackEmpty1(S)

```
if top1(S) = 0 then
    return TRUE
else
    return FALSE
end if
```

StackEmpty2(S)

```
if top2(S) = length(S) + 1 then
    return TRUE
else
    return FALSE
end if
```

StackFull1(S)

if $top1(S) = top2(S) - 1$ **then**
    return TRUE
else
    return FALSE
end if

StackFull2(S)

if $top1(S) = top2(S) - 1$ **then**
    return TRUE
else
    return FALSE
end if

Push1(S,x)

if $StackFull1(S)$ **then**
    error OVERFLOW
else
    $top1(S) \leftarrow top1(S) + 1$
    $S[top1(S)] \leftarrow x$
end if

Push2(S,x)

if $StackFull2(S)$ **then**
    error OVERFLOW
else
    $top2(S) \leftarrow top2(S) - 1$
    $S[top2(S)] \leftarrow x$
end if

Pop1(S)

```
if StackEmpty1(S) then
    error UNDERFLOW
else
    x ← S[top1[S]]
    top1(S) ← top1(S) − 1
    return x
end if
```

Pop2(S)

```
if StackEmpty2(S) then
    error UNDERFLOW
else
    x ← S[top2[S]]
    top2(S) ← top2(S) + 1
    return x
end if
```

Coda (queue)

La struttura di dati **coda** rappresenta un insieme dinamico nel quale le operazioni di inserimento e cancellazione seguono una politica

FIFO = First In First Out

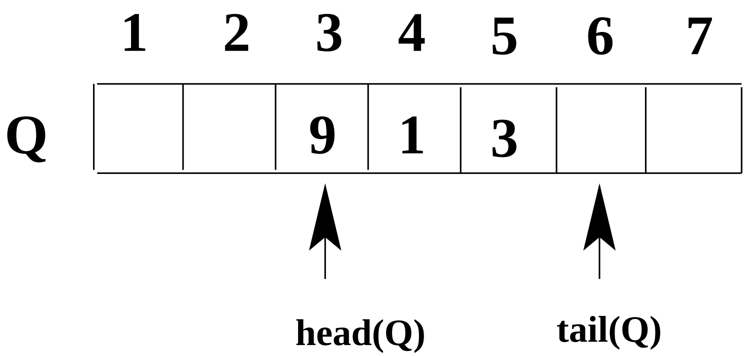
Ciò significa che l'oggetto cancellato è quello che è stato inserito **per primo**.

Coda (queue)

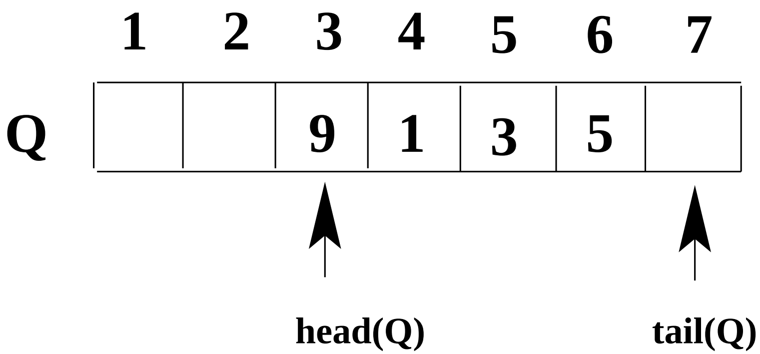
Una coda Q possiede un attributo $head(Q)$ che contiene un puntatore all'**inizio** della coda (dove escono gli elementi), un attributo $tail(Q)$ che contiene un puntatore alla **fine** della coda (dove entrano gli elementi), e le seguenti quattro operazioni:

- $QueueEmpty(Q)$ che controlla se la coda Q è vuota;
- $QueueFull(Q)$ che controlla se la coda Q è piena;
- $Enqueue(Q, x)$ che inserisce l'oggetto x alla fine della coda Q;
- $Dequeue(S)$ che rimuove l'oggetto all'inizio della coda Q.

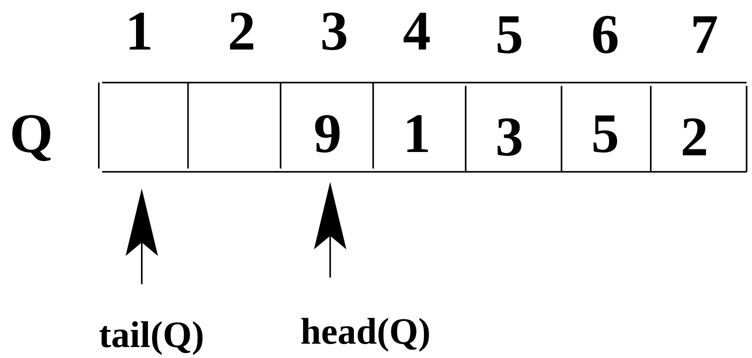
Una coda Q



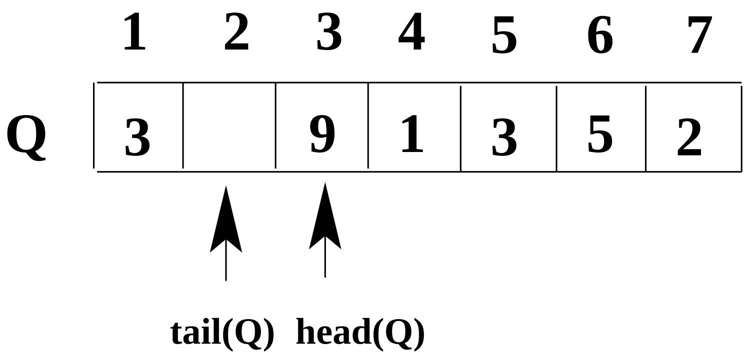
$Enqueue(Q, 5)$



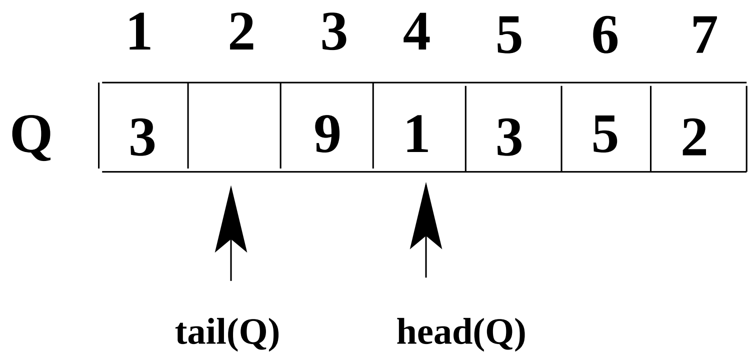
$Enqueue(Q, 2)$



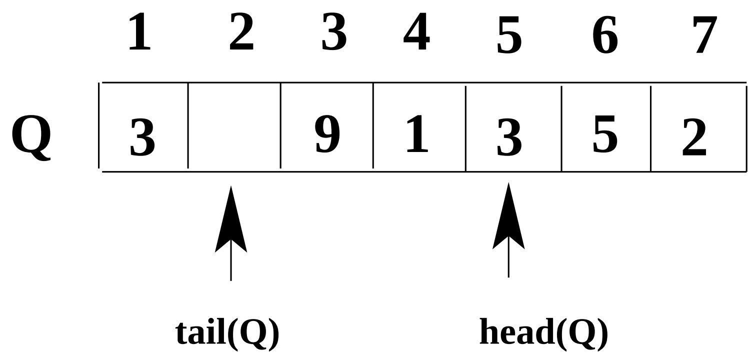
$Enqueue(Q, 3)$



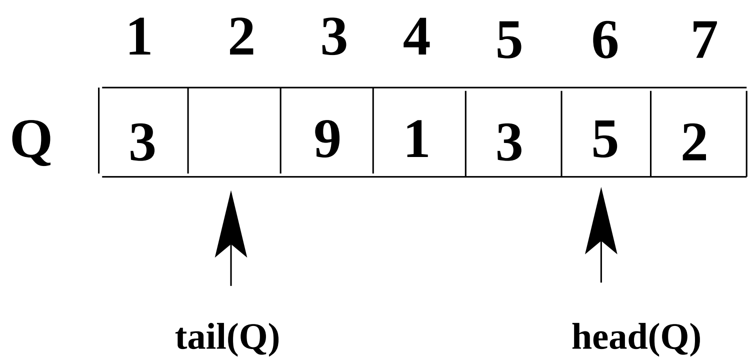
$Dequeue(Q)$



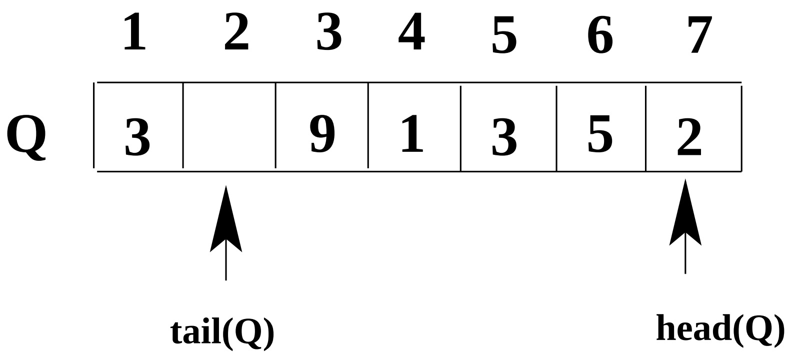
$Dequeue(Q)$



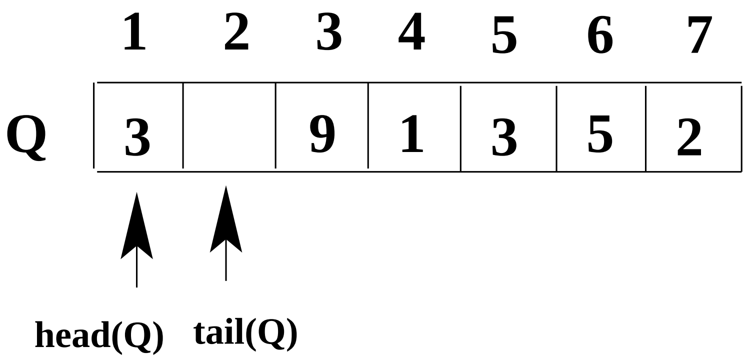
$Dequeue(Q)$



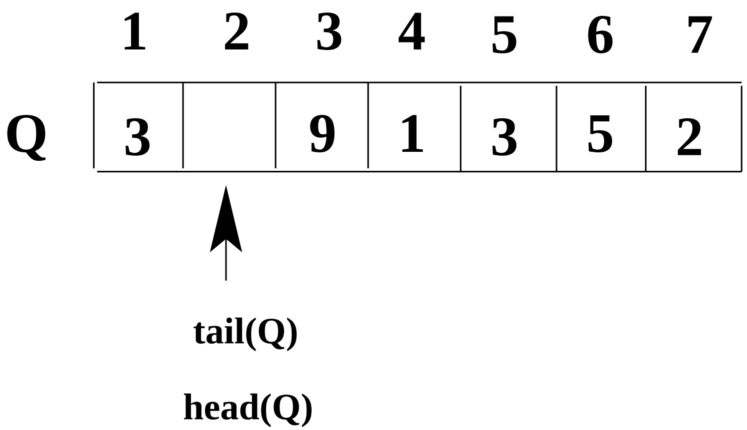
$Dequeue(Q)$



$Dequeue(Q)$



$Dequeue(Q)$



QueueEmpty(Q)

```
if head(Q) = tail(Q) then
    return TRUE
else
    return FALSE
end if
```

QueueFull(Q)

```
if ((tail(Q) + 1) mod length(Q)) = head(Q) then
    return TRUE
else
    return FALSE
end if
```

Enqueue(Q,x)

if $QueueFull(Q)$ **then**
  error OVERFLOW
else
  $Q[tail(Q)] \leftarrow x$
  $tail(Q) \leftarrow (tail(Q) + 1)\ mod\ length(Q)$
end if

Dequeue(Q)

if $QueueEmpty(S)$ **then**
  error UNDERFLOW
else
  $x \leftarrow Q[head(Q)]$
  $head(Q) \leftarrow (head(Q) + 1)\ mod\ length(Q)$
  return x
end if

$\boxed{\textbf{Cosa fa?}}$

while not $QueueEmpty(P)$ **do**
  $Enqueue(Q, Dequeue(P))$
end while

Esercizio *Sia Q una coda. Si scriva una procedura $QueueLength(Q)$ che ritorna la lunghezza della coda Q.*

Esercizio *Sia Q una coda. Si scriva una procedura $QueueLength(Q)$ che ritorna la lunghezza della coda Q.*

QueueLength(Q)

if $head(Q) \leq tail(Q)$ **then**
    return $tail(Q) - head(Q)$
else
    return $length(Q) - head(Q) + tail(Q)$
end if

Esercizio *Sia Q una coda. Si scriva una procedura $QueueLength(Q)$ che ritorna la lunghezza della coda Q.*

QueueLength(Q)

if $head(Q) \leq tail(Q)$ **then**
  return $tail(Q) - head(Q)$
else
  return $length(Q) - head(Q) + tail(Q)$
end if

La complessità di $QueueLength(Q)$ è costante, cioè $\Theta(1)$.

Esercizio *Sia Q una coda. Si scriva una procedura*

$$QueueReverse(Q)$$

che inverte la coda Q usando una pila e una procedura

$$StackReverse(S)$$

che inverte la pila S usando una coda.

QueueReverse(Q)

```
S ← ∅
while not QueueEmpty(Q) do
    Push(S, Dequeue(Q))
end while
while not StackEmpty(S) do
    Enqueue(Q, Pop(S))
end while
```

StackReverse(S)

```
Q ← emptyset
while not StackEmpty(S) do
    Enqueue(Q, Pop(S))
end while
while not QueueEmpty(Q) do
    Push(S, Dequeue(Q))
end while
```