In questa lezione

- **Code di priorità**

[CLRS10] cap. 6, par. 5

Coda di priorità: cos'è?

- **Una coda di priorità è una struttura dati che permette di gestire dei dati con chiave numerica, la loro priorità.**

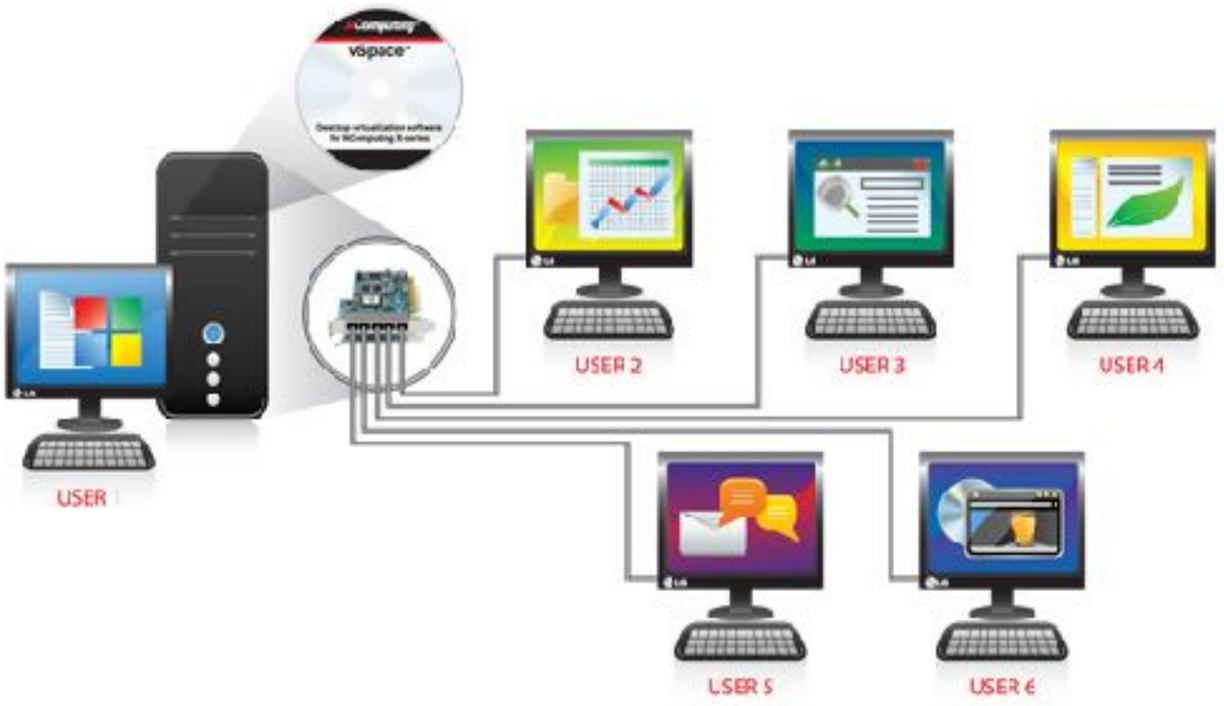
- **In una coda di priorità la modalità di prelievo si basa sulla priorità associata agli elementi, mentre in una coda la modalità di prelievo è basata sull'ordine di inserimento: il primo inserito è il primo a essere estratto**

Coda di max-priorità: le operazioni

- **Una coda di max-priorità offre le operazioni di**
  - **Insert(S,x): inserisce x in S**
  - **Maximum(S): restituisce l'elemento con priorità massima, senza estrarlo**
  - **Extract-Max(S): restituisce l'elemento con priorità massima eliminandolo dall'insieme**
  - **Increase-key(S,x,k): cambia la priorità di x al valore k, se k è maggiore o uguale di quella presente**

Applicazioni

In caso di condivisione di un computer fra più utenti: viene svolto per primo il lavoro con più alta priorità



Coda di min-priorità: le operazioni

- **Una coda di min-priorità offre le operazioni di**
  - **Insert(S,x): inserisce x in S**
  - **Minimum(S): restituisce l'elemento con priorità minima**
  - **Extract-Min(S): restituisce l'elemento con priorità minima e lo rimuove dall'insieme**
  - **Decrease-key(S,x,k): porta a k la priorità di x, se k è minore di quella presente**

Applicazioni

Simulazione guidata da eventi: Gli elementi della coda sono quelli da simulare e ad ogni elemento è associato il tempo nel quale si può verificare.

Implementazione: caso max-priorità

- **Per implementare una coda con priorità si utilizza un heap binario, in cui sono memorizzate le priorità**
- **Si utilizza un handle, aggancio (un puntatore, un intero), per legare la priorità nell'array a un elemento e viceversa (in tal caso l'handle dell'elemento è l'indice nell'array in cui compare la sua priorità).**

Implementazione: caso max-priorità

Usando un semplice array per rappresentare le priorità, si ottengono le seguenti prestazioni:

implementazione	insert	Max	Extract-Max
array non ordinato	Θ(1)	Θ(n)	Θ(n)
array ordinato crescente	Θ(n)	Θ(1)	Θ(1)

Un max-heap consente un'implementazione efficiente per tutte queste operazioni!

Le operazioni

- **Abbiamo visto come implementare l'estrazione del massimo e l'inserimento in O(lg n), se n sono gli elementi della coda.**
- **consideriamo ora l' operazione di aumento di una chiave.**

- **Avremo così completata l'implementazione delle operazioni fondamentali, alle quali poi aggiungeremo anche la cancellazione e il join, cioè la fusione di due max-Heap in un unico Max-heap.**

Aumento di una chiave

- **Può essere necessario aumentare la priorità di un elemento in corso d'opera.**
- **L'aumento della chiave può portare a una violazione della proprietà del max-heap, naturalmente nei confronti del padre.**

Increase-Key passo 1

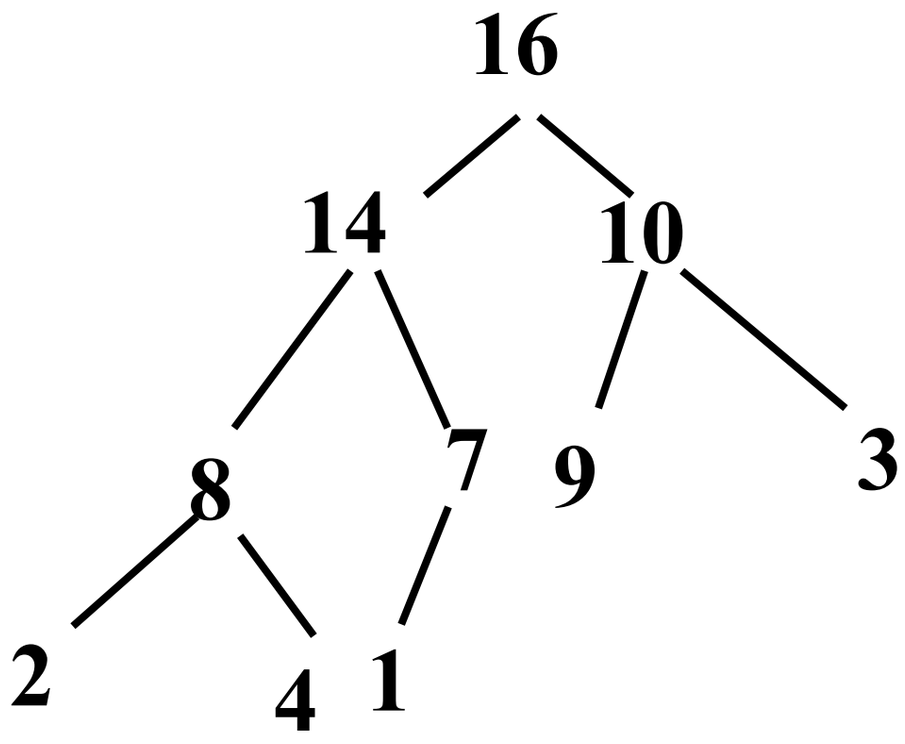
A=

16	14	10	8	7	9	3	2	4	1	4	0	7
1	2	3	4	5	6	7	8	9	10	11	12	13

Increase-key(A,9,15)
passo 1

A=

16	14	10	8	7	9	3	2	15	1	4	0	7
1	2	3	4	5	6	7	8	9	10	11	12	13



4 ⇨ 15

15

Increase-Key passo 2

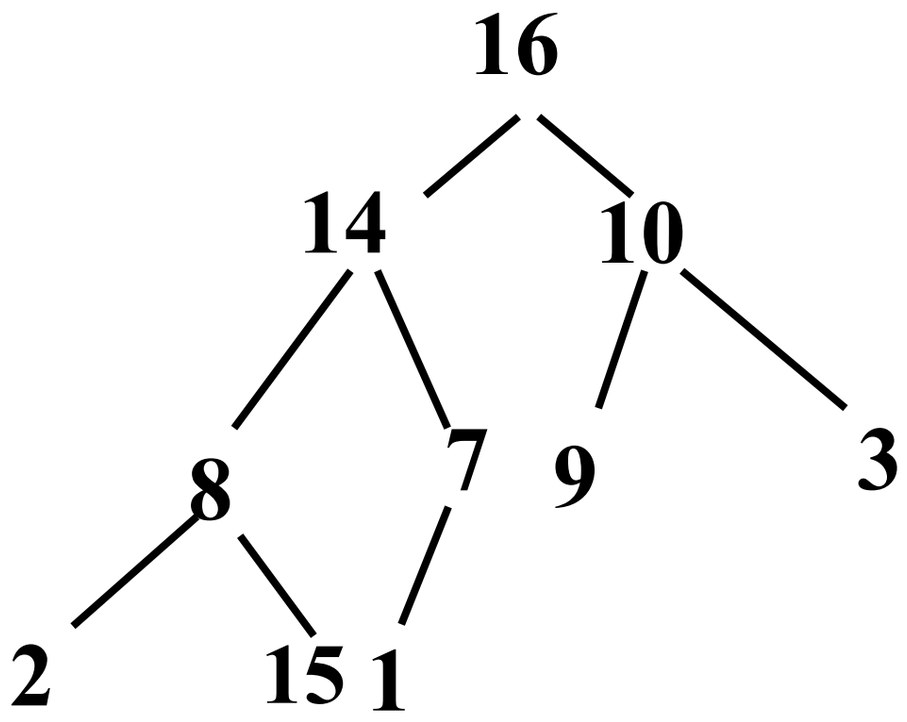
A=

16	14	10	8	7	9	3	2	15	1	4	0	7
1	2	3	4	5	6	7	8	9	10	11	12	13

Increase-key(A,9,15)
passo 2

A=

16	15	10	14	7	9	3	2	8	1	4	0	7
1	2	3	4	5	6	7	8	9	10	11	12	13



pseudocodice Increase-key

Heap-increase-key(A,i,key)

prec: A è un max-heap e 1≤i≤A.heap-size

postc: sostituisce key ad A[i], se key ≥ A[i] e ripristina la proprietà del max-heap

if key < A[i] then error "la nuova chiave è più piccola"

A[i] = key

while (i > 1 and A[i/2] < A[i])

L'array A[1...heap-size] soddisfa la proprietà del max-heap tranne al più nella posizione i

scambia A[i/2] con A[i]

i = i/2 "si risale al padre di A[i]"

Increase-key: analisi

Heap-increase-key(A,i,key)

▷ A è un max-heap. Sostituisce key ad A[i], se key ≥ A[i] e ripristina la proprietà del max-heap

```
if key < A[i] then error "la nuova chiave è più piccola"
A[i] = key
   while (i > 1 and A[i/2] < A[i])
         scambia A[i/2] con A[i]
         i = i/2
```

Caso peggiore: Θ(lg n),
n = heap-size(A)

Sommario dei costi

Complessità nel caso peggiore di diverse implementazioni di una coda di max-priorità con n elementi

implementazione	insert	Max	Extract-Max
array non ordinato	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
array ordinato in ordine crescente	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
heap	$\Theta(\lg n)$	$\Theta(1)$	$\Theta(\lg n)$

Riassumendo

La MAX-HEAPIFY, che ha complessità logaritmica nel numero degli elementi, è il metodo chiave per mantenere la proprietà del max-heap.

I metodi
MAX-HEAP-INSERT,
HEAP-EXTRACT-MAX,
HEAP-INCREASE-KEY, e
HEAP-MAXIMUM,
tutti di complessità al più logaritmica nel numero degli elementi, permettono l'implementazione efficiente di una coda di priorità con il max-heap.

Cancellare un elemento

Può essere inoltre utile l' operazione

Delete(A,i): cancella A[i] da A

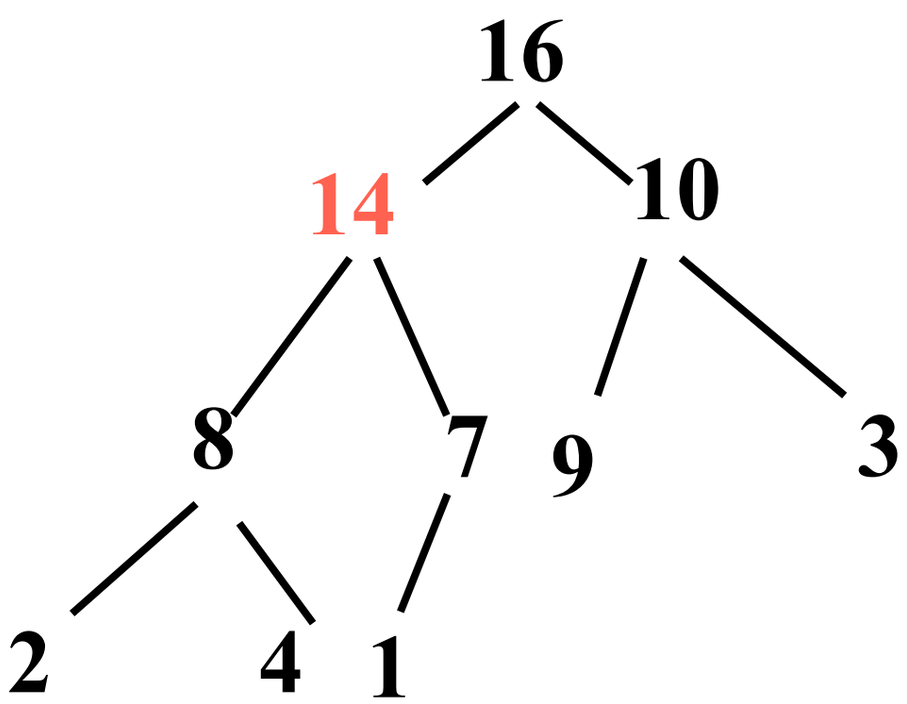
delete(A,i) caso 1 passo 1

Si vuole cancellare A[2] = 14

A=

16	14	10	8	7	9	3	2	4	1	4	0	7
1	2	3	4	5	6	7	8	9	10	11	12	13

A.heap-size=10



14 viene sostituito dall'ultimo elemento.

In questo caso la proprietà è violata rispetto ai figli:

usiamo la Max-Heapify

delete(A,i) caso 1 passo 2

Si vuole cancellare A[2] = 14

A=

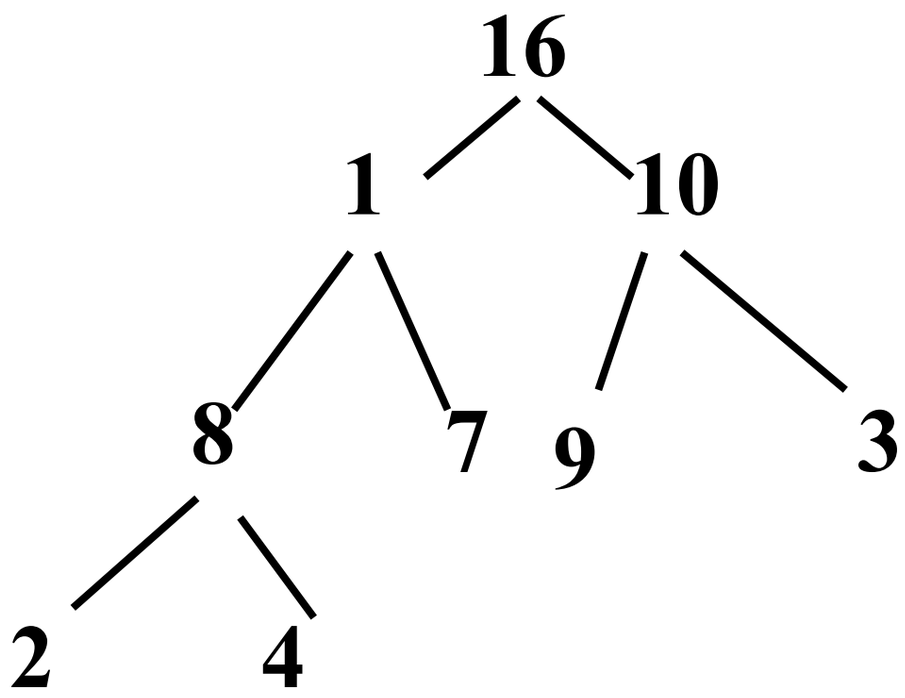
16	14	10	8	7	9	3	2	4	1	4	0	7
1	2	3	4	5	6	7	8	9	10	11	12	13

Heap-delete(A,2)

A.heap-size=10

A=

16	8	10	4	7	9	3	2	1	1	4	0	7
1	2	3	4	5	6	7	8	9	10	11	12	13



A.heap-size=9

Max-Heapify per ripristinare la proprietà del Max-Heap

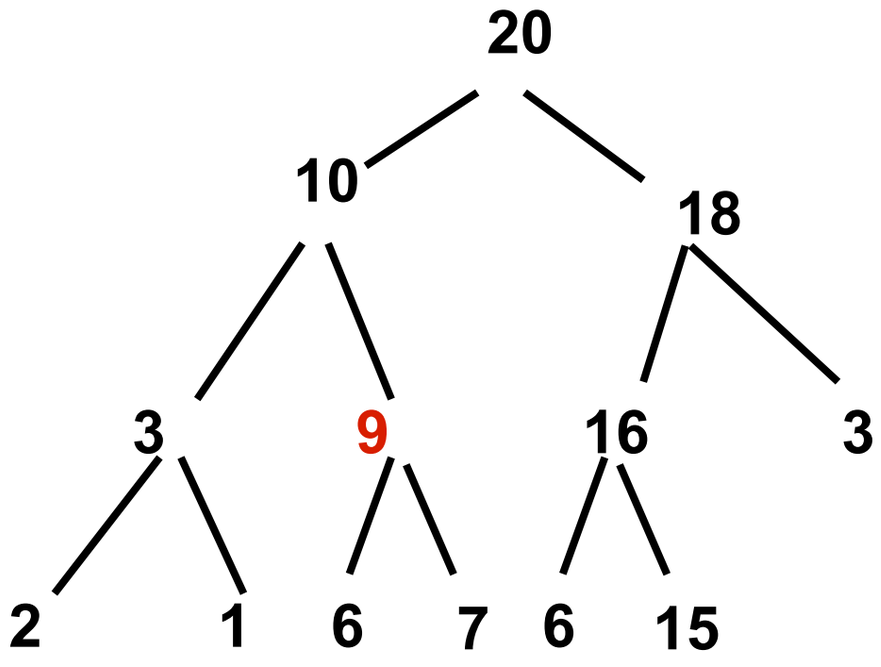
delete(A,i) caso 2 passo 1

Si vuole cancellare A[5] = 9

A=

20	10	18	3	9	16	3	2	1	6	7	6	15
1	2	3	4	5	6	7	8	9	10	11	12	13

heap-size(A)=13



9 viene sostituito dall'ultimo elemento.

La proprietà è violata rispetto al padre.

delete(A,i) caso 2 passo 2

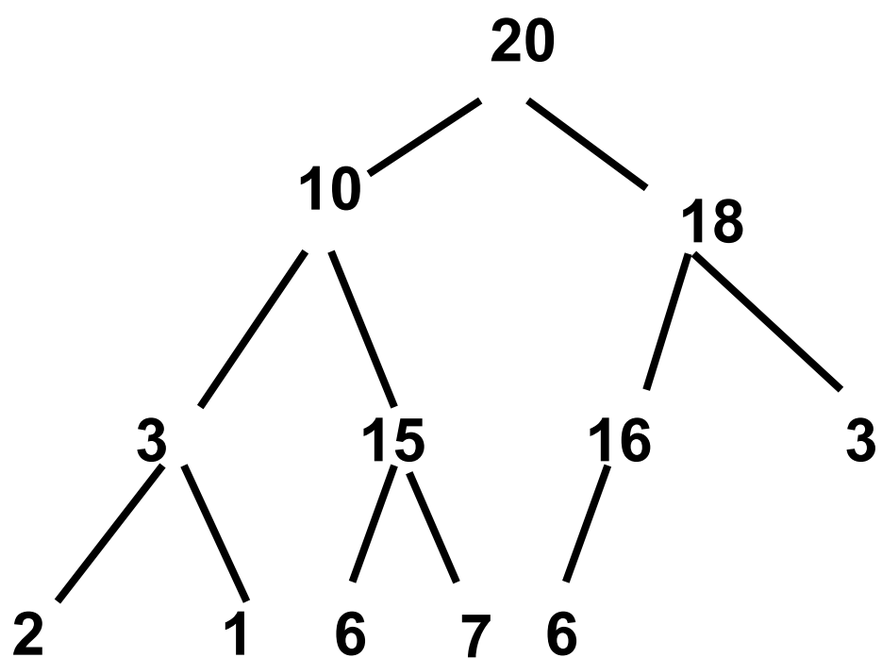
Si vuole cancellare A[5] = 9

A=

20	10	18	3	9	16	3	2	1	6	7	6	15
1	2	3	4	5	6	7	8	9	10	11	12	13

Heap-delete(A,5)

heap-size(A)=13



A=

20	15	18	3	10	16	3	2	1	6	7	6	15
1	2	3	4	5	6	7	8	9	10	11	12	13

heap-size(A)=12

Per ripristinare la proprietà del max-heap bisogna scambiare la chiave con quella del padre

delete(A,i) pseudocodice

Max-Heap-delete(A,i)

prec: A è un max-heap, 1≤i≤A.heap-size

postc: A è il max-heap in input privato dell'i-simo elemento

```
A[i]  = A[A.heap-size]
A.heap-size =  A.heap-size-1
if (2i+1 ≤ A.heap-size and  A[i] < A[2i+1])
  or (2i ≤ A.heap-size and A[i] < A[2i])
        then Max-Heapify(A,i)
```

proprietà violata rispetto ai figli

```
while (i > 1 and A[i] > A[i/2])
```

proprietà violata rispetto al padre

```
    do scambia A[i] con A[i/2]
       i = i/2
```

delete(A,i) analisi

Max-Heap-delete(A,i)

prec: A è un max-heap, 1≤i≤A.heap-size

postc: A è il max-heap in input privato dell'i-simo elemento

A[i] = A[A.heap-size]

A.heap-size = A.heap-size-1

if (2i+1 ≤ A.heap-size and A[i] < A[2i+1])

or (2i ≤ A.heap-size and A[i] < A[2i])

then Max-Heapify(A,i)

proprietà violata rispetto ai figli

while (i > 1 and A[i] > A[i/2]) **proprietà violata rispetto al padre**

do scambia A[i] con A[i/2]

i = i/2

Caso peggiore: Θ(lg n),

n = heap-size(A)

Join

Dati due Max-Heap come posso costruire un Max-Heap che contiene gli elementi di entrambi?