HEAP mucchio

Struttura dati per implementare

CODE CON PRIORITÀ

CODA
semplice
FIFO
First In First out

inserzione in fondo $\Theta(1)$
estrazione del primo elemento $\Theta(1)$

CODA CON PRIORITÀ

- inserzione
- estrazione dell'elemento a priorità max
- lettura dell'el. a priorità max

	array non ordinato	array ordinato	lista non ordinata
inserzione	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
estrazione max	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
lettura max	$\Theta(n)$	$\Theta(1)$	"

HEAP

inserzione	$O(\log n)$
estrazione del max	$O(\log n)$
lettura del max	$\Theta(1)$

1) Albero binario completo addossato a sinistra

$n = 2^h - 1$ se $n \neq 2^{h-1}$

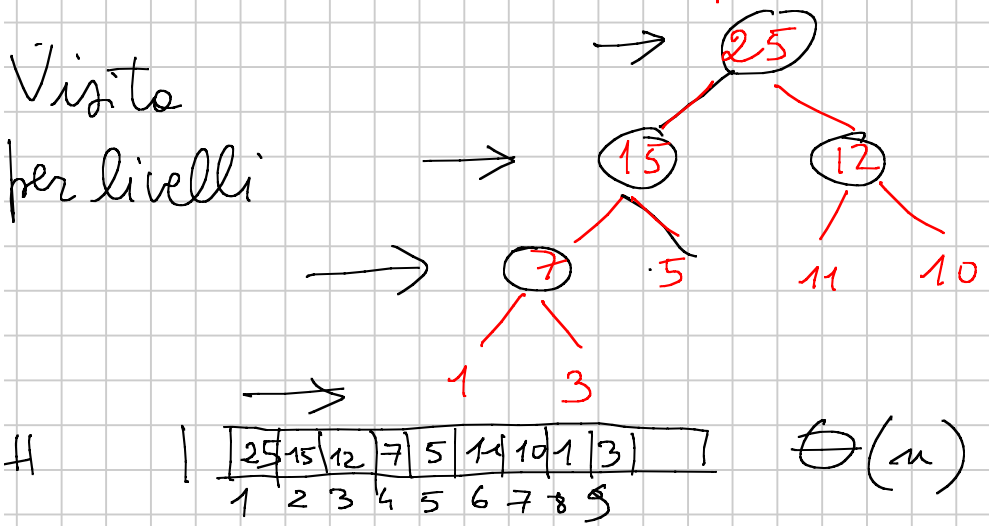
$h-1$ livelli dell'albero sono completi

ultimo livello: tutte le foglie sono addossate a sinistra

invariante dell'Heap

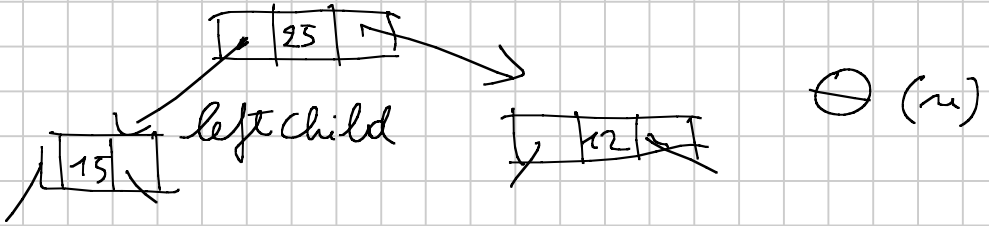
2) Max-heap (Min heap)

Per ogni nodo dell'heap: la chiave del nodo è maggiore (minore per Min-heap) delle chiavi contenute nei figli



MAX-HEAP

ricerca $O(n) \cong$ ricerca seq.



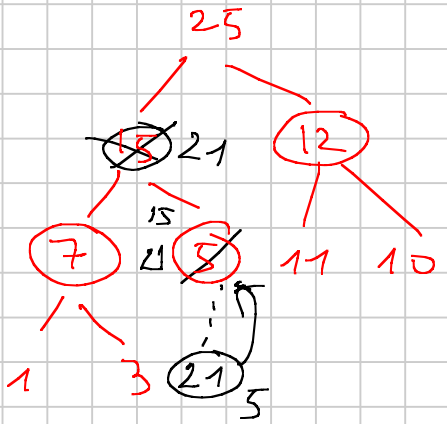
H

1	2	3	4	5	6	7	8	9	10
25	15	12	7	5	11	10	1	3	21

$\forall i$ leftchild(i) è in posizione $2i$ se esiste

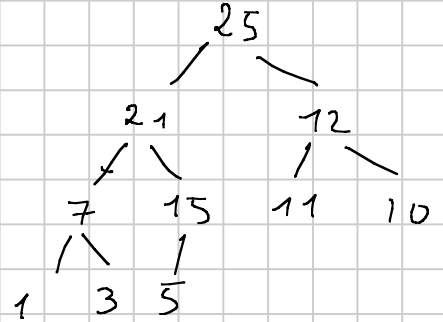
rightchild(i) " $2i+1$

parent(i) $\lfloor i/2 \rfloor$



Coda di priorità

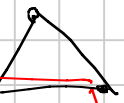
inserzione 21



inserzione in un Heap di size n

$O(\log n)$

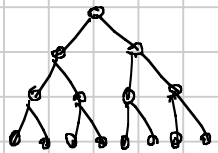
per Heap di altezza h



$$2^h \leq n < 2^{h+1}$$

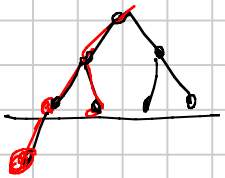
$h = 3$

numero max di nodi



$h = 3 \quad n = 2^4 - 1 = 15$

numero minimo di nodi



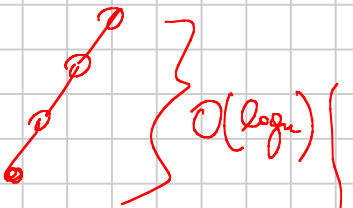
$n = 2^h$

$$h \leq \log n < h + 1$$

$$h = \lfloor \log n \rfloor \begin{cases} h \leq \log n \\ h > \log n - 1 \end{cases}$$

Heap ha altezza $\Theta(\log n)$ $\quad n$ = numero di nodi

```
MAX_HEAP_INSERT (A, key);
A.heapsize = A.heapsize + 1;
A[A.heapsize] = -∞;
HEAP_INCREASE_KEY (A, A.heapsize, key);
```




```
HEAP_INCREASE_KEY (A, i, key)
if (key < A[i]) "errore";
A[i] = key;
while ((i > 1) && (parent(i) < A[i])) {
    scambia A[i] ⇔ A[parent];
    i = parent(i);
}
```

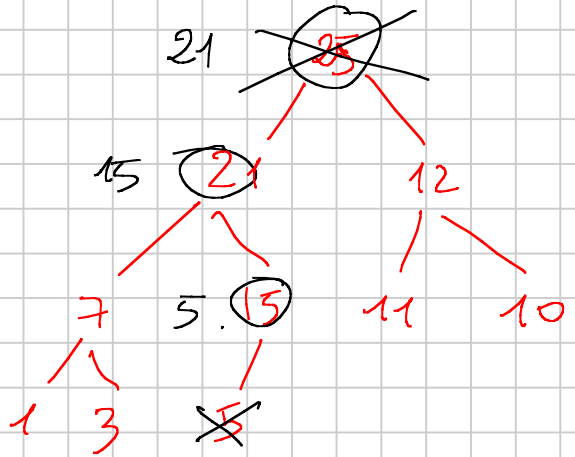
Θ(1)

input: Heap di size A.heapsize con in posizione i una chiave fuori posto perché più piccola degli antenati.

output → MAX-heap

Titolo nota 17/03/2016

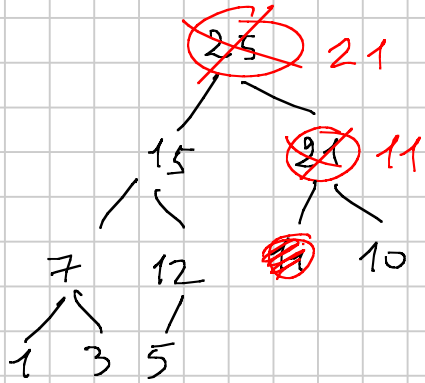
ESTRAZIONE DEL MAX

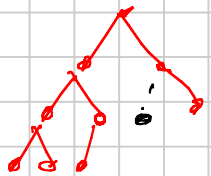


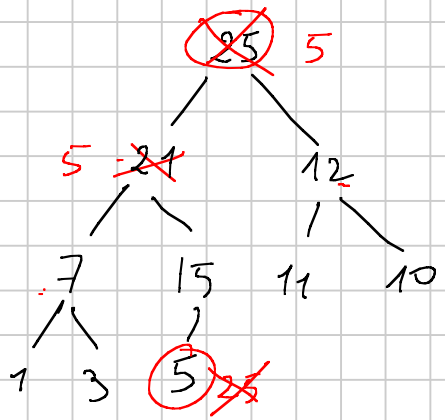
A

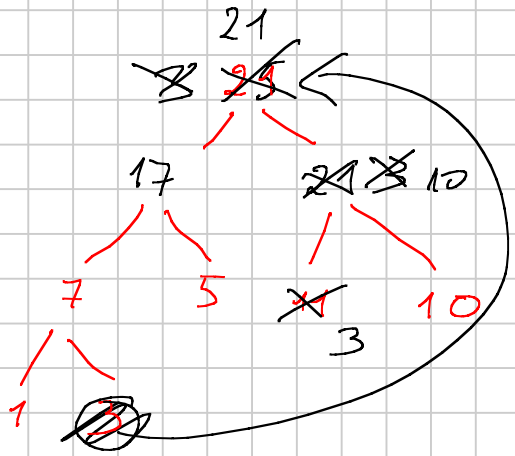
A.heapsize = 10

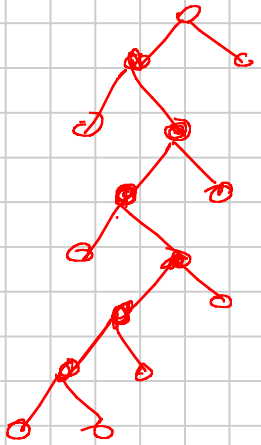
25	21	12	7	15	11	10	1	3	5









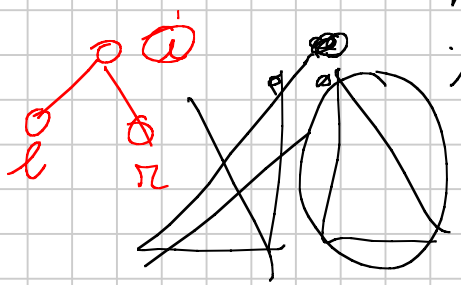


$\Theta(1)$

Estrazione max

$O(\log n)$

1) Scambio radice con l'ultima foglia
2) ristrutturo la proprietà MAX-HEAP confrontando la radice con i suoi 2 figli e con via finché non trovo il suo posto



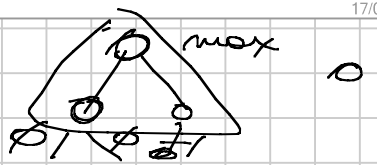
$$\begin{cases} T(n) = \Theta(1) & n \leq 3 \\ T(n) \leq T\left(\frac{n}{2}\right) + \Theta(1) \end{cases}$$

$T(n) = \Theta(\log n)$

```
HEAP-EXTRACT-MAX (A)
if A.heapsize < 1   "errore underflow"
max = A[1];
A[1] = A[heapsize];
MAX-HEAPIFY (A, 1);
return max;
```

Input: Max-heap con un elemento in posizione i eventualmente più piccolo dei suoi discendenti

$n = 3$




```
MAX-HEAPIFY (A, i):
l = leftchild (i);
r = rightchild (i);
if (l ≤ A.heapsize && A[l] > A[i]
    max = l;
    else max = i;
if (r ≤ A.heapsize && A[r] > A[max]
    max = r;
if (max ≠ i) scambia A[i] ↔ A[max]
MAX-HEAPIFY (A, max);
```

- Costruzione Heap
- HEAP_SORT