Lezione 26 marzo

Thursday, March 26, 2020 9:05 AM

http://repl.it

Code semplice: inserzioni in coda PUSH
Estrazione in testa POP

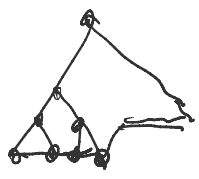
Code con priorità - Heap

1) struttura
2) informazione nei nodi

1) Heap è un albero binario completo sbilanciato a sinistra

n = numero di nodi dell'albero

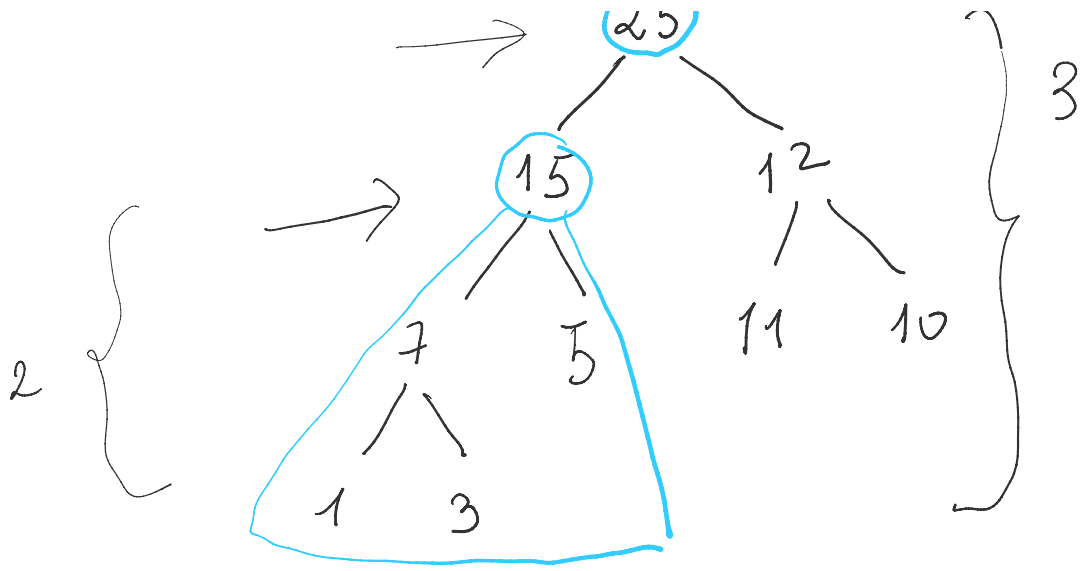
$n = 2^h - 1 \Rightarrow$ 

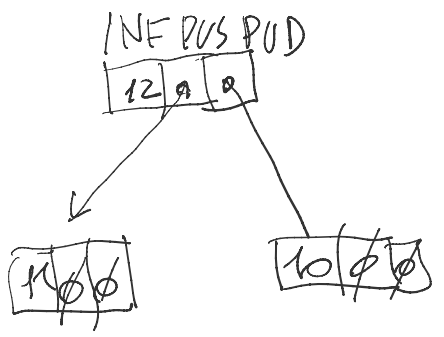
$n \neq 2^h - 1 \Rightarrow$  incompleto

2) informazione e = (nome, priorità)

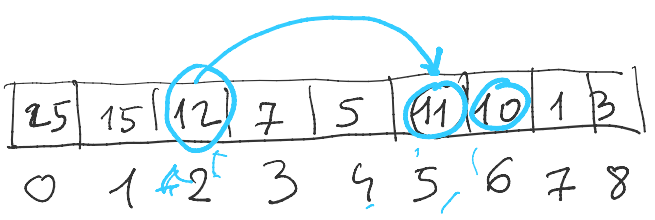
Max heap: per tutti i nodi
ogni elemento è maggiore di tutti quelli contenuti nel sottalbero di cui è radice.
Nella radice è contenuto il max

Min heap





Struttura heap → array



heaparray

nodo in posizione i heaparray[i]

sinistro (i) in posizione $2i+1$

destro (i) " $2i+2$

padre (i) in " $\left\lfloor \frac{i-1}{2} \right\rfloor$ ricorsione

---

Relazione tra n = numero di nodi
h = altezza dell'albero

altezza di un nodo = max distanza (calcolata come numero di archi) tra il nodo e una foglia

altezza di un albero = altezza della radice.

Th: in albero binario completo ($\triangle$) di altezza h:

$N_h = 2^{h+1} - 1$ nodi

$F = 2^h$ foglie

$N_h = 2^h - 1$ nodi

$F_h = 2^h$ foglie

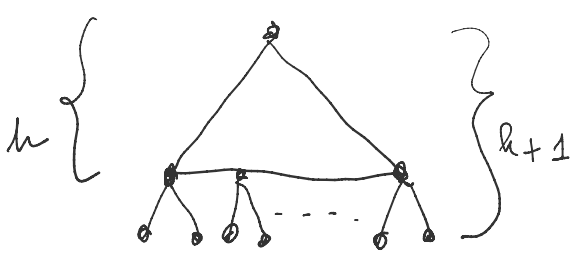
$I_h = 2^h - 1$ nodi interni

Prova ; induzione su h

base $h = 0 \Rightarrow$ •

$N_0 = 2^1 - 1 = 1$
$F_0 = 2^0 = 1$
$I_0 = 2^0 - 1 = 0$
} vero

passo induttivo $h \to h+1$



$$N_{h+1} = N_h + 2 \cdot F_h = 2^{h+1} - 1 + 2 \cdot 2^h$$
$$= 2^{h+2} - 1$$

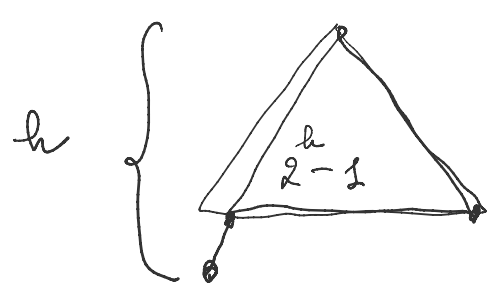
$$F_{h+1} = 2 \cdot 2^h = 2^{h+1}$$

$$I_{h+1} = 2^{h+1} - 1$$

Un albero completo di n nodi ha altezza $\log(n+1) - 1$

$$n = 2^{h+1} - 1 \qquad n+1 = 2^{h+1}$$
$$h = \log(n+1) - 1$$

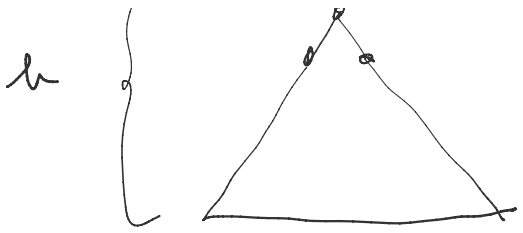
Studiamo la rel. tra h e n in uno heap



heap con minimo numero di nodi

$$n = 2^h$$

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



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

$h \leq \log_2 n < h+1$

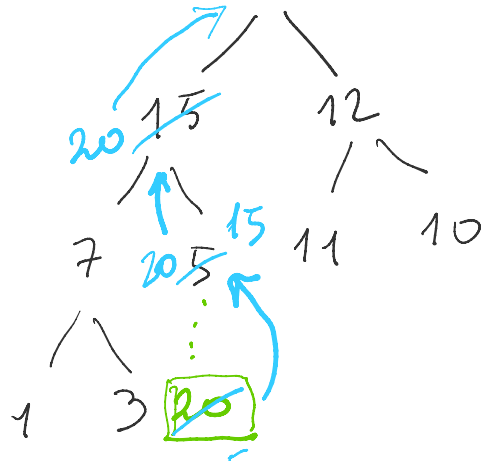
$\begin{cases} h \leq \log n \\ h > \log n - 1 \end{cases}$ $\quad h = \lfloor \log n \rfloor$

$h = \Theta(\log n)$

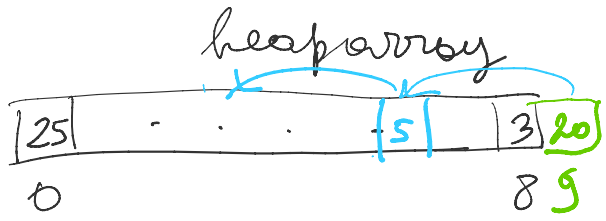
Operazioni

1) Empty ? $\Theta(1)$
2) Max ? $\Theta(1)$
3) Estrazione Max
4) Inserzione

Heap è strutturato per realizzare efficientemente le istruzioni 2/3/4.



heapsize = 9

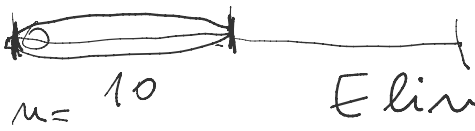


e con priorità 20

numero di confronti max = percorso foglia radice

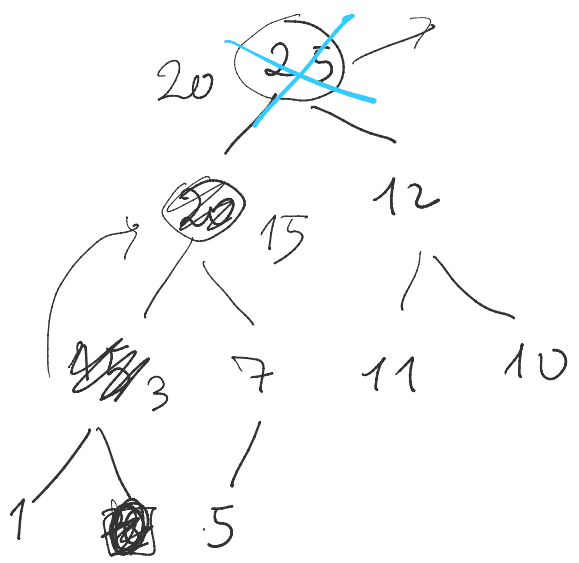
Enqueue (e); $\quad e$ = priorità

- Verifica Raddoppio ();
  heaparray [heapsize] = e;
  heapsize ++;
  Riorganizza heap (heapsize - 1);



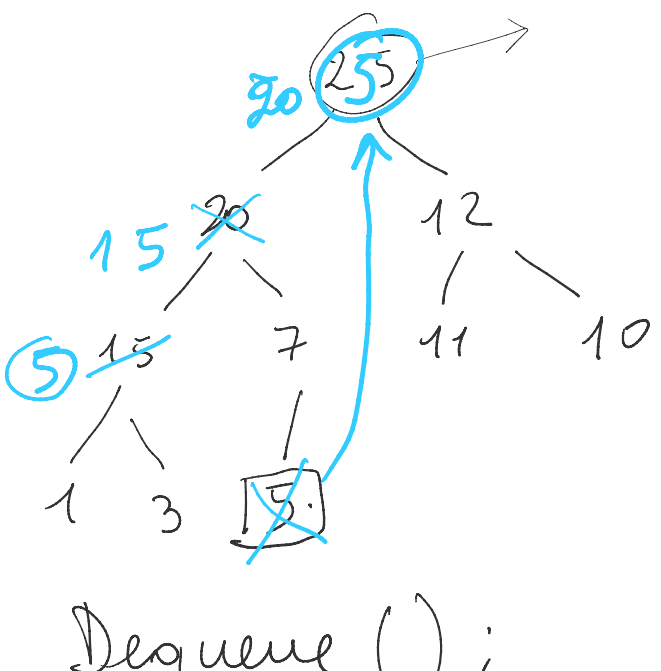
Eliminazione del massimo

heapsize 10



Dequeue

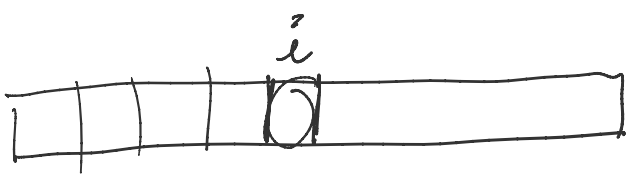
max = heaparray[0]




```
Dequeue ();
if (!Empty()){
    max = heaparray[0];
heapsize--;  heaparray[0] = heaparray[heapsize-1];
             RiorganizzaHeap (0);
             VerificaDimensionamento ();
             return max;
}
```

RiorganizzaHeap (i)

input



heaparray

input 

heap dove un solo el. non rispetta le proprietà

output → heap

```
Riorganizzaheap(i):
//bottom-up//
while (i>0 && heaparray[i] >
        heaparray[Padre[i]]) {
    Scambia (i, padre(i));
    i = padre(i);
}
//top down//
while (sinistro(i) < heapsize &&
        i =! MigliorePadreFigli(i)) {
    migliore = Migliorepadre figli(i)
    Scambia (i, migliore);
    i = migliore;
}
```

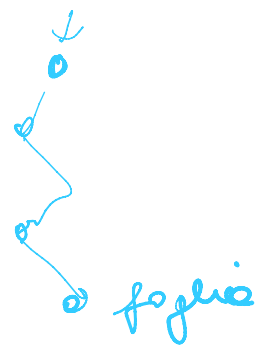
inserzione

$O(\log n)$

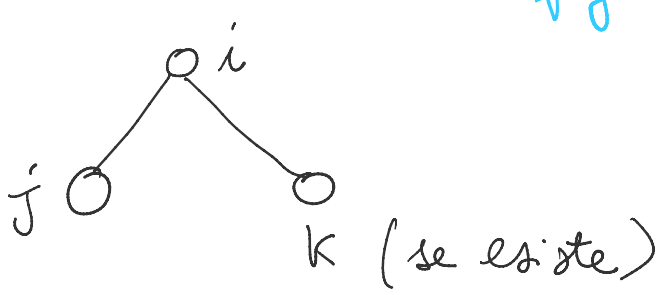
$O(\log n)$

eliminazione

$\Theta(1)$



Migliorepadre figli(i)



1) Empty $\Theta(1)$
2) max $\Theta(1)$
3) Eliminazione max $O(\log n)$
4) Inserzione $O(\log n)$

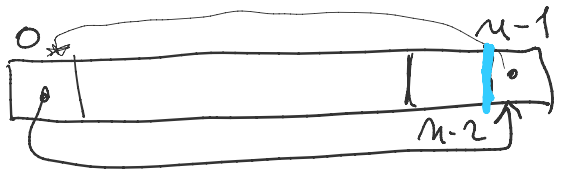
Code con priorità

Code con priorità

Possiamo usare l'heap per fare il Sorting!

HeapSort

1) Costruisci un max-heap di n elementi
2) for Scambia la radice con l'ultimo el (heaparray[heapsize-1])



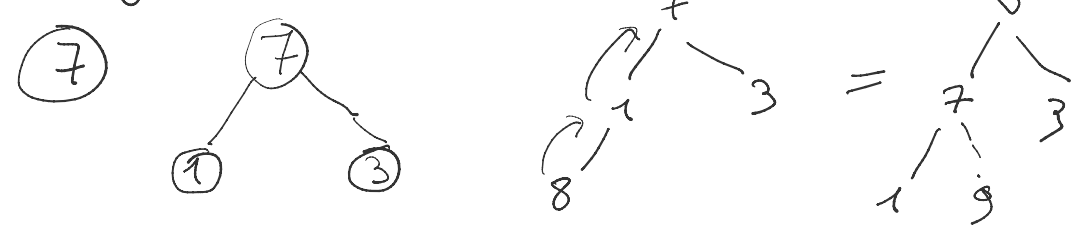
heap

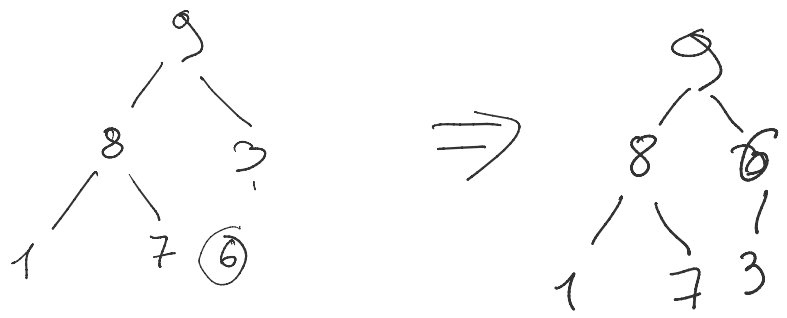
Riorganizzaheap(.,0);

3) Riorganizza heap

7	1	3	8	9	6
0	1	2	3	4	5

Inserisco gli el. in un heap inizialmente vuoto





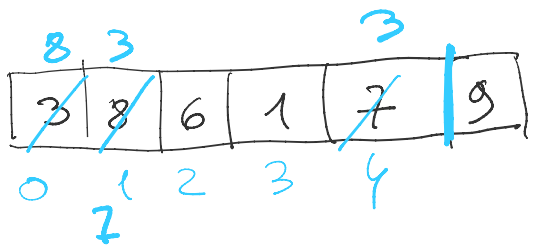
heaparray

9	8	6	1	7	3
0	1	2	3	4	5

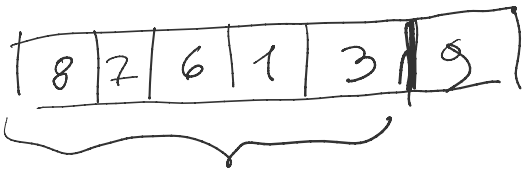
costruire un heap?

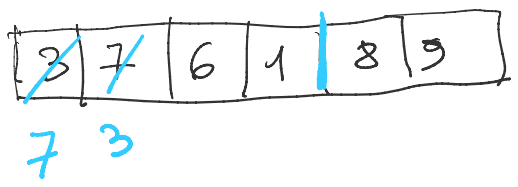
n operaz. di inserzione

1) $O(n \log n)$

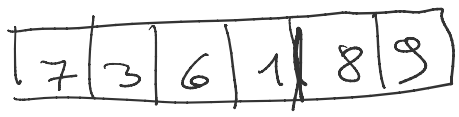


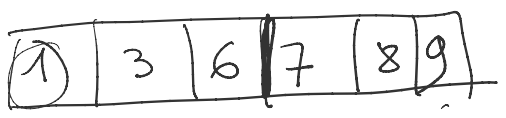
Riorg. heap(0)





Riorgheap(0)





Riorg. heap (0)

```
HeapSort (heaparray):
    heapsize = 0;
O(n log n) for (i=0; i<n; i++) Enqueue(heaparray(i));
    while (heapsize > 0) {
        Scambia (heaparray[0], heaparray(heapsize-1));
        heapsize--;
        Riorganizaheap(0); O(log n)
    }
```

O(n log n)

HeapSort in loco { O(n log n) tempo; senza spazio aggiuntivo }