

### Esercizio

Dato A vettore di interi di lunghezza  $m$  e dato  $K$  intero  
decidere se esistono due elementi di A  $x$  e  $y$  tali che  
 $x + y = K$

Soluzione  $\Theta(m \log m)$

### Esercizio

Dato A vettore --- decidere se in A ci sono elementi  
ripetuti

Soluzione  $\Theta(m \log m)$

### Esercizio

In un vettore A di interi chiamiamo Majority Candidate  
un elemento che compare in A più di  $\lceil \frac{m}{2} \rceil$  volte

Potrebbe non esistere

Descrivere un algoritmo che dato A

- se A ha un Majority Candidate x restituisci x
- se A non ha " " restituisci NULL

Esiste una soluzione tempo  $\Theta(m)$  in-place

# Coda con Priorità (Capitolo 6)

Heap

Coda

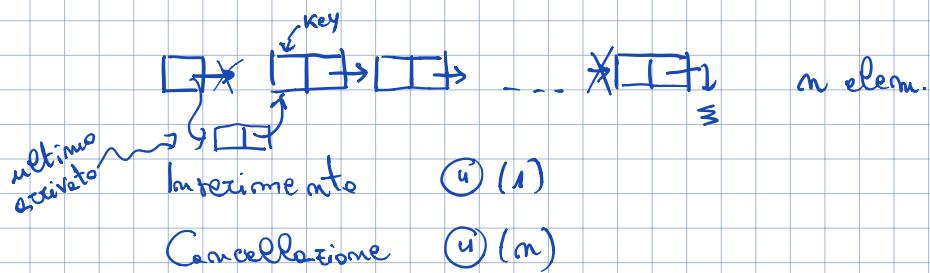
Struttura Dati Astratta

Sequenziale, Inserimenti e Cancellazioni

gestiti FIFO  
}

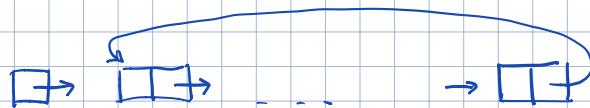
Implementazioni Coda

- Liste Concatenate

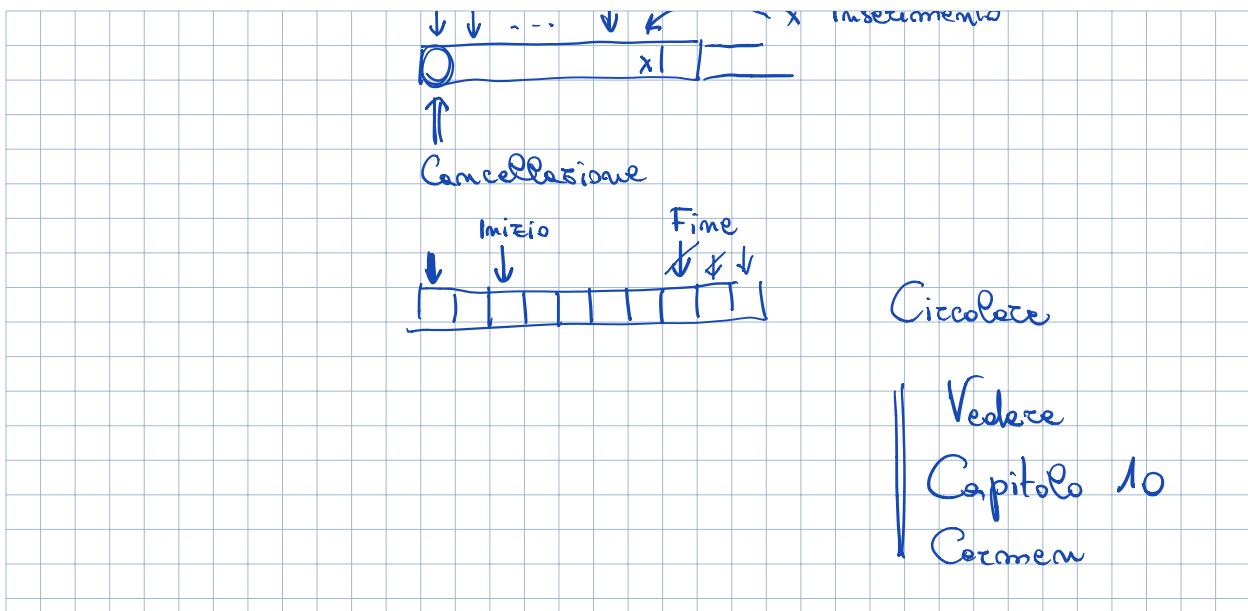


Inserimento in coda      }  
Cancellazione in testa      }  $\Theta(1)$

- Liste circolari



- Vettore Sorpasso Dimensionato



## Code con Priorità

Strutture Dati Astratte

Ad ogni elemento è associata una priorità

Si estroie per primo l'elemento con priorità più alta

Operazioni : Inserimento

Cancellazione

Modifica delle priorità

{  
\*}

Heap

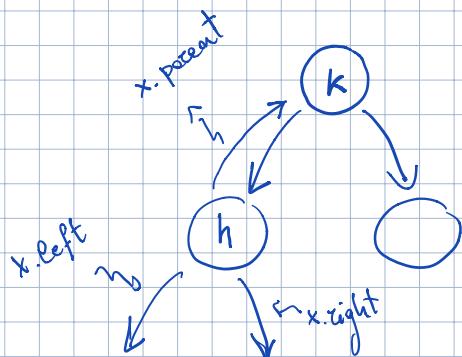
Implementazione delle Code con Priorità

con costo  $\mathcal{O}(\log n)$  per le 3 operazioni (\*)

Def

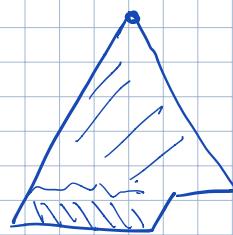
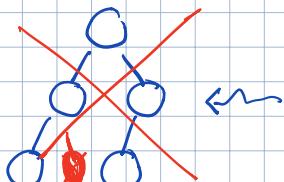
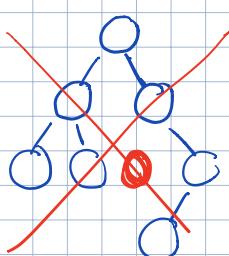
Una Max-heap è un albero binario quasi-completo  
in cui ogni nodo ha una chiave intera e vale che  
( $x$  nodo     $x.key$      $x.left$      $x.right$      $x.parent$ )

$$\forall x \quad x \neq \text{root} \quad x.key \leq x.parent.key$$

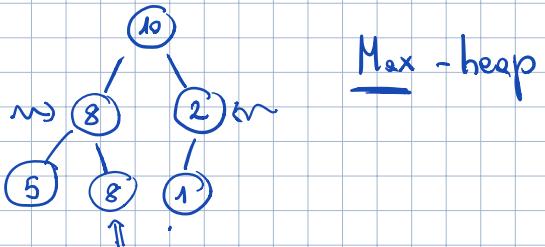


Albero binario quasi-completo

Albero binario completo fino al penultimo livello  
Ultimo livello riempito da sx verso dx



## Esempio



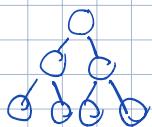
Max - heap

Min - heap scambiando  $\leq$  con  $\geq$

## Proprietà

- La radice ha priorità massima
- La priorità minima si trova in una foglia
- n numeri di modi  $\Rightarrow$  l'altezza  $\Theta(\log n)$

$\therefore$



$$2^0, 2^1, 2^2, \dots, 2^h$$

$$m = \sum_{i=0}^h 2^i = 2^{h+1} - 1$$

$$h+1 = \log_2(m+1) \quad h = \log_2(m+1) - 1 \\ = \Theta(\log m)$$



$$m = \sum_{i=0}^{h-1} 2^i + 1 = 2^h - 1 + 1 = 2^h$$

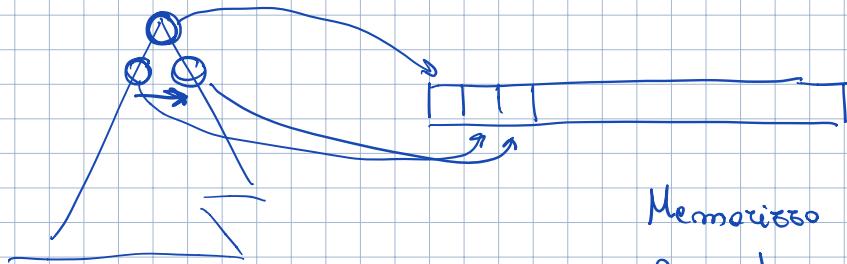
$$h = \log_2 m \quad \Theta(\log m)$$

$$\log_b m = \Theta(\log_c m) \quad b, c > 1$$



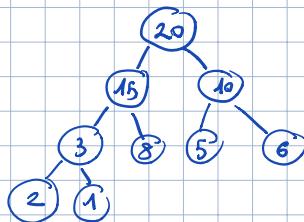
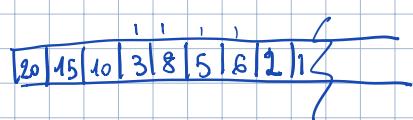
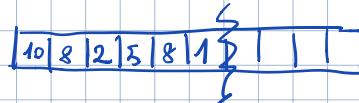
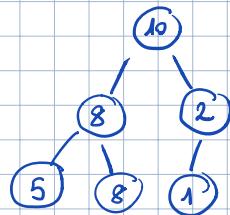
$$h = 0$$

Implementazione delle Mex-heap con vettori sovrapposti.



Memorizzo gli elementi nel vettore  
scendendo l'albero  
per livelli e partire  
dalle radici e procedendo  
in ogni livello da sx  
verso dx

Esempio

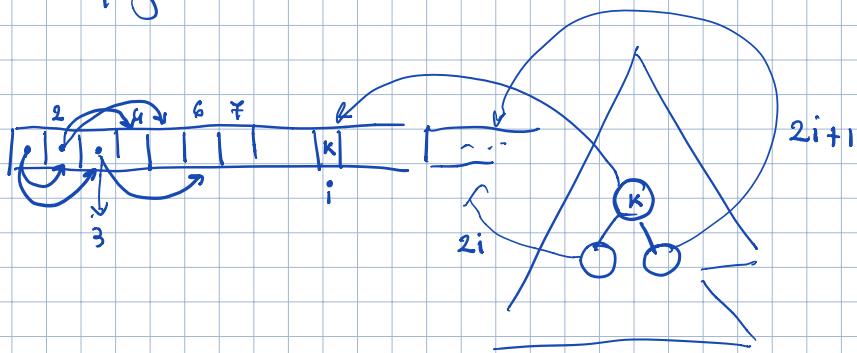


Una max-heap sarà gestita come un vettore  $H$  di lunghezza  $H.length$  in cui la heap occupa la porzione  $[1..H.heapsize]$

$$0 \leq H.heapsize \leq H.length$$

Prop. Se  $H$  è un vettore che memorizza una Max-heap ---

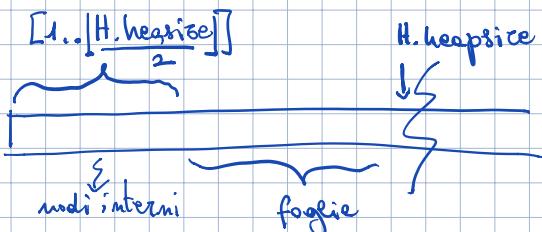
- $H[i]$  è il massimo di  $H$
- Gli elementi da  $H.heapsize/2$  ad  $H.heapsize$  sono "foglie"



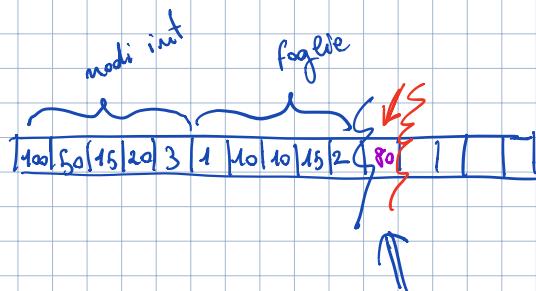
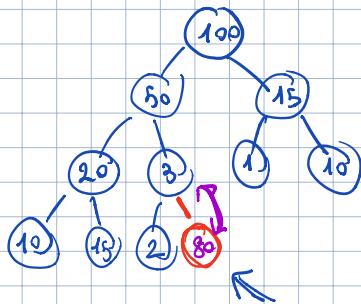
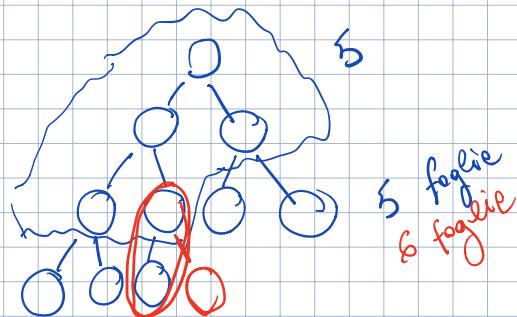
Se  $2 \cdot j > H.heapsize$  allora  $H[j]$  è una foglia

Perché  $H[j]$  non sia una foglia  $2j \leq H.heapsize$

$$j \leq \frac{H.heapsize}{2}$$



Esempio



Inserimento di una nuova chiave

- Incremento H. heapsize
- Inserisco in posizione H. heapsize
- Confronto con il genitore ed eventualmente scambio fino a che non sistemo il problema

Left (i) {

    return 2i

}

Right (i) {

    return 2i+1

}

Parent (i) {

    return  $\frac{i}{2}$

```

MaxHeapInsert ( H, k ) {
    if ( H.heapsize < H.length ) {
        H.heapsize ← H.heapsize + 1
        H[H.heapsize] ← k
        i ← H.heapsize
        while ( H[i] > H[Parent(i)] ) {
            Swap ( H, i, Parent(i) )
            i ← Parent(i)
        }
    }
}

```

Véritable Rie. ?  
Complexité ?