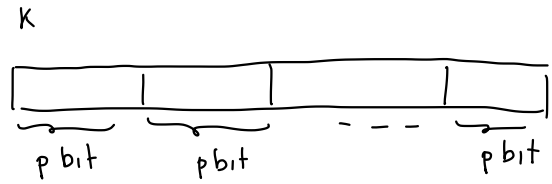
Algoritmi e Strutture Dati (15/11/2021)

ESERCIZIO (Anagramma)

$h(k) = k \bmod m$

se $m = 2^p - 1$



se prendo k, k' ottenute l'una dall'altra permutando le parole di p bit

$$h(k) = h(k')$$

$\rightarrow 0 .. 2^p - 1 \rightsquigarrow$ cifra in base 2^p

OSSERVAZIONE: dato un numero

$m = a_k a_{k-1} \dots a_1 a_0$ base b

allora $m \bmod b-1 = \left(\sum_{i=0}^{k} a_i \right) \bmod (b-1)$

[divisibilità per 9

$a_k \dots a_0$ divisibile per 9

sse

$a_k \dots a_0 \bmod 9 = 0$

sse

$\underbrace{\sum_{i=0}^{k} a_i}_{} \bmod 9 = 0$

ovvero è divisibile per 9]

per induzione su k

$(k=0)$ $m = a_0 \cdot b^0 = a_0$

$\overset{a_0}{m} \bmod b-1$

$\parallel$

$\underbrace{\sum_{i=0}^{0} a_i}_{a_0} \bmod b-1$

$(K \to K+1)$ $\quad n = a_{K+1} \underbrace{a_K \cdots a_1 a_0}_{n'}$

$$= a_{K+1} \cdot b^{K+1} + n'$$

$$n \bmod (b-1) = (a_{K+1} b^{K+1} + n') \bmod (b-1)$$

$$(x+y) \bmod z = (x \bmod z + y \bmod z) \bmod z$$

$$= \left(a_{K+1} \bmod (b-1) * \overbrace{(b \bmod (b-1))}^{1}{}^{K+1} + \underbrace{n' \bmod (b-1)}_{\left(\sum_{i=0}^{K} a_i\right) \bmod (b-1)} \right) \bmod (b-1)$$

$$= \left(a_{K+1} \bmod (b-1) + \left(\sum_{i=0}^{K} a_i\right) \bmod (b-1) \right) \bmod (b-1)$$

$$= \left(\sum_{i=0}^{K+1} a_i\right) \bmod (b-1)$$

(<u>più</u>: vale in generale sostituendo $b-1$ con c t.c. $b \bmod c = 1$)

<u>ESERCIZIO</u> (ABR- Inorder)

Dato albero binario T

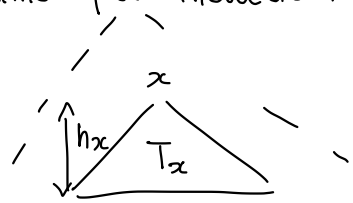
T è ABR sse Inorder(T.root) produce i nodi ordinati in modo crescente

$\boxed{\text{dim}}$

dim. più in generale

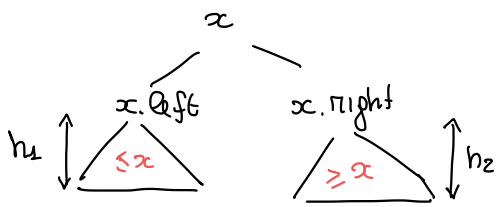
sottoalbero radicato in x è ABR sse InOrder(x) produce la lista dei nodi del sottalbero radicato in x crescente per chiave

($\Rightarrow$) procediamo per induzione sull'altezza h_x dell'albero radicato in x



($h_x = 0$) T_x è vuoto, ovvero $x = nil$ $\rightsquigarrow$ ok, banalmente vero

($h_x > 0$) allora



$h_1, h_2 < h_x$

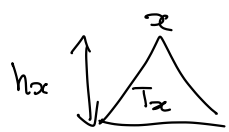
InOrder (x) produce

ordinata

ordinata $\leq$ x $\leq$ ordinata $\rightarrow$ x ipotesi induttiva

$\underbrace{x_1 \dots x_k}_{\text{InOrder}(x.left)}$ $\quad$ $\underbrace{y_1 \dots y_{k'}}_{\text{InOrder}(x.right)}$

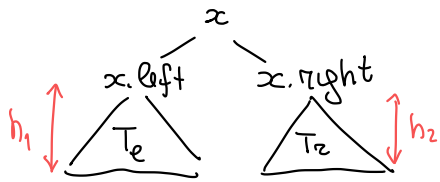
($\Leftarrow$) se InOrder (x) produce la lista dei nodi ordinata $\Rightarrow$ x è radice di ABR



induzione su h_x

($h_x = 0$) $x = nil$, albero radicato in x è vuoto $\rightsquigarrow$ è ABR

($h_x > 0$) allora



$h_1, h_2 < h_x$

InOrder (x) $\quad \underbrace{\underbrace{x_1 \dots x_k}_{\text{InOrder}(x.left)} \leq x}_{\text{ordinata per ipotesi}} \; \overbrace{y_1 \dots y_{k'}}^{\text{InOrder}(x.right)}$

$\hookrightarrow$ $x_1 \dots x_k$, $y_1 \dots y_{k'}$ ordinate $\rightarrow$ T_l, T_r sono ABR (ip. induttiva)

$\hookrightarrow$ T_e, T_r sono ABR

$T_e \leq x$

$T_r \geq x$

$\Rightarrow$ T_x è ABR

$\square$

ESERCIZIO : (Max Heap Ordinato)

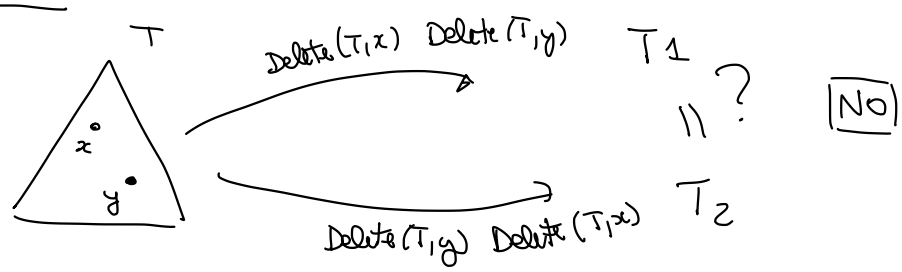
dato max-heap elencare i suoi elementi in ordine crescente (o decrescente) in tempo $O(n)$

$\rightarrow$ A $\rightsquigarrow$ max-heap in tempo $\Theta(n)$

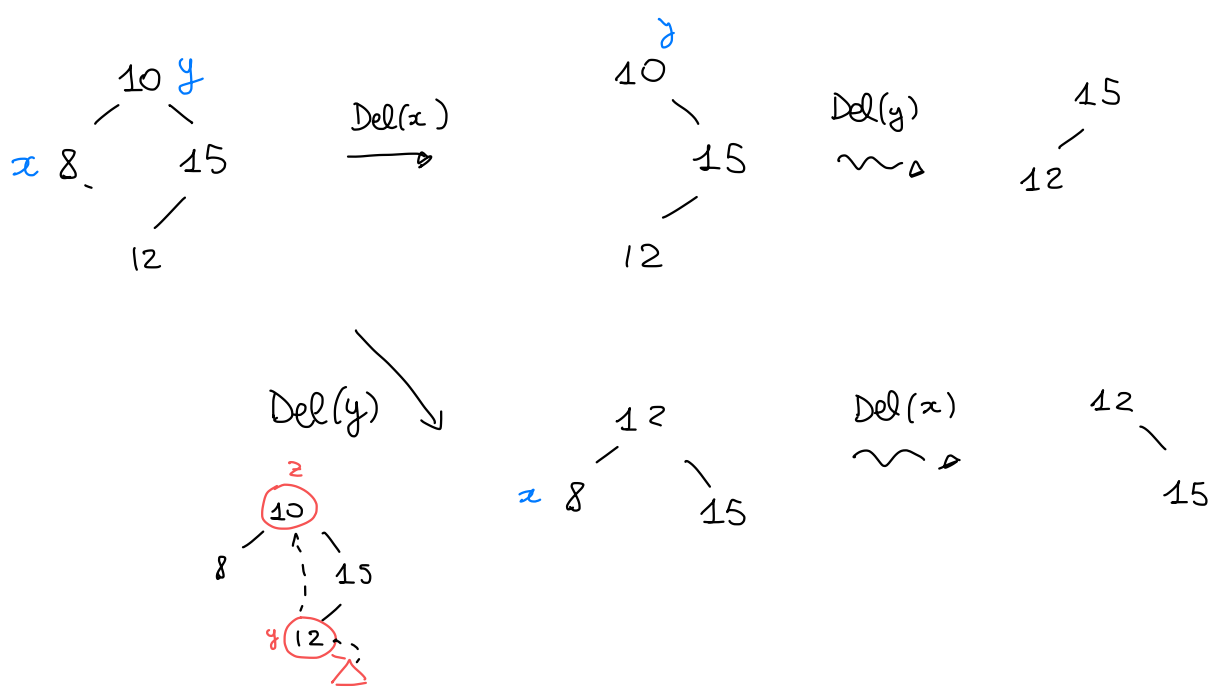
$\rightarrow$ se esistesse quello che è stato richiesto, potrei ordinare in tempo lineare (impossibile: ordinamento è $\Omega(n \log n)$)

ESERCIZIO : costruire ABR in tempo lineare : impossibile.

ESERCIZIO (DELETE)



es:



ESERCIZIO (Insert Ricorsiva)

```
InsertRec ( x , z , parent)
    if x = nil
        z.p = parent
        return z
    else
        if z.key < x.key
            x.left  =  InsertRec (x.left , z , x)
        else
            x.right =  InsertRec ( x.right , z , x)

        return x
```

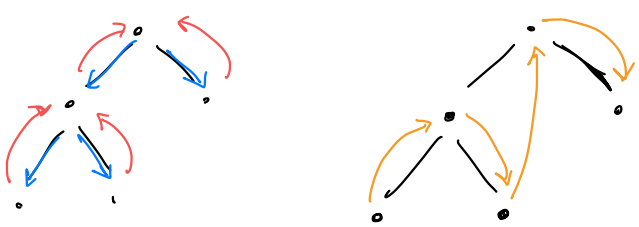
→ inserisce z nel sottoalbero radicato in x

→ ritorna la radice dell'albero in cui l'elemento z è inserito

```
Insert (T, z)
    Insert (T.root , z , nil)
```

ESERCIZIO : (ABR successore)

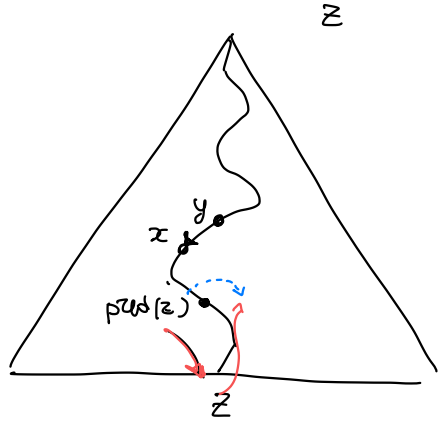
ABR x $x.left$
$x.right$
~~$x.p$~~ $x.succ$



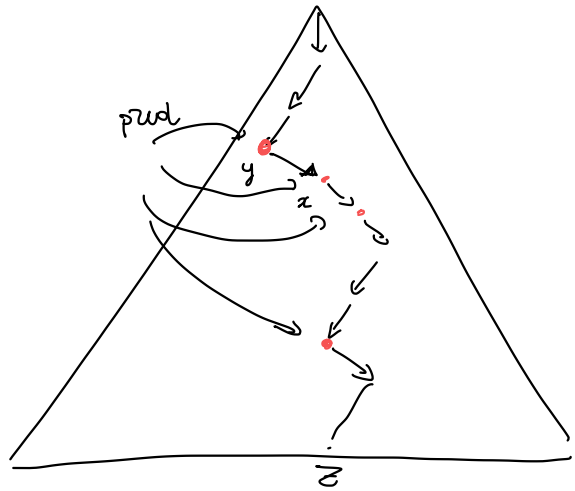
successore facile $O(1)$ (invece che $O(h)$)

massimo, minimo, search invariate $O(h)$

* Insert (T, z)

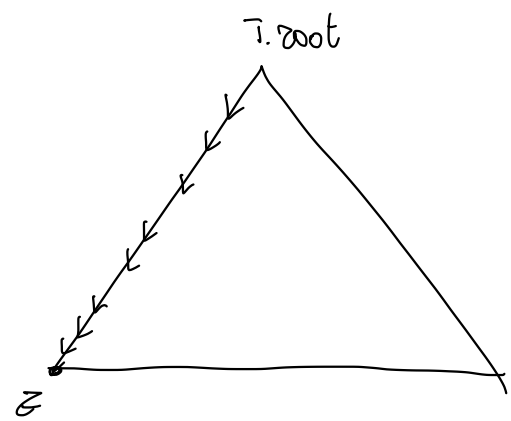


Idea :



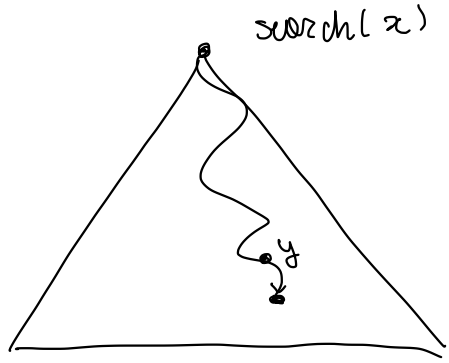

```
Insert (T, z)
    x = T.root
    y = nil
    pred = nil
    while x ≠ nil
        y = x
        if z.key < x.key
            x = x.left
        else
            x = x.right
            pred = y
    if y = nil
        T.root = z
    else if z.key < y.key
            y.left = z
        else
            y.right = z

    if pred ≠ nil
        z.succ = pred.succ
        pred.succ = z
    else
        z.succ = y
```

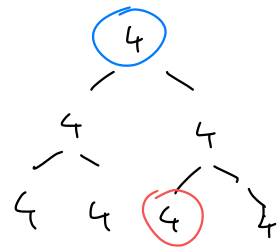
$O(h)$



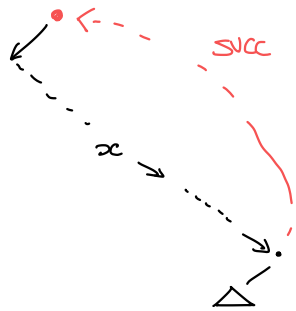
Parent

Parent (T, x)





come fare ?

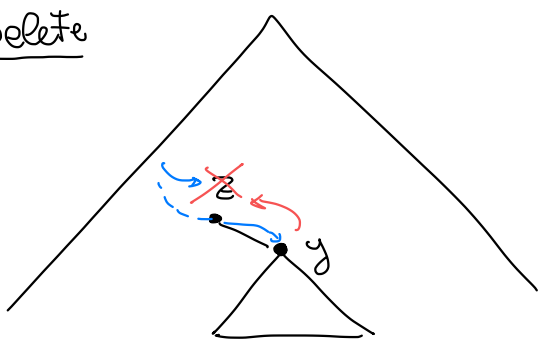


costo $O(h)$

* <u>Predecessore</u> : implementazione "std" con parent calcolato

$O(h^2)$

* <u>Delete</u>



$O(h^2)$