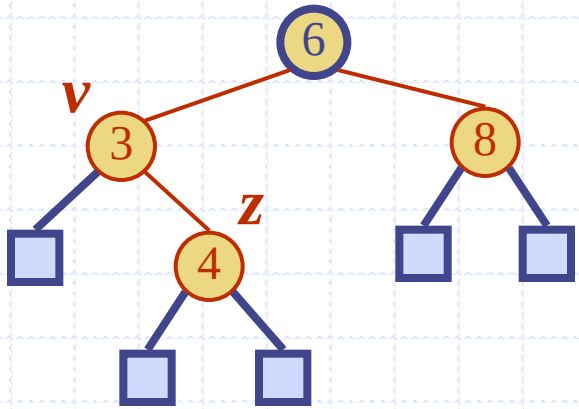
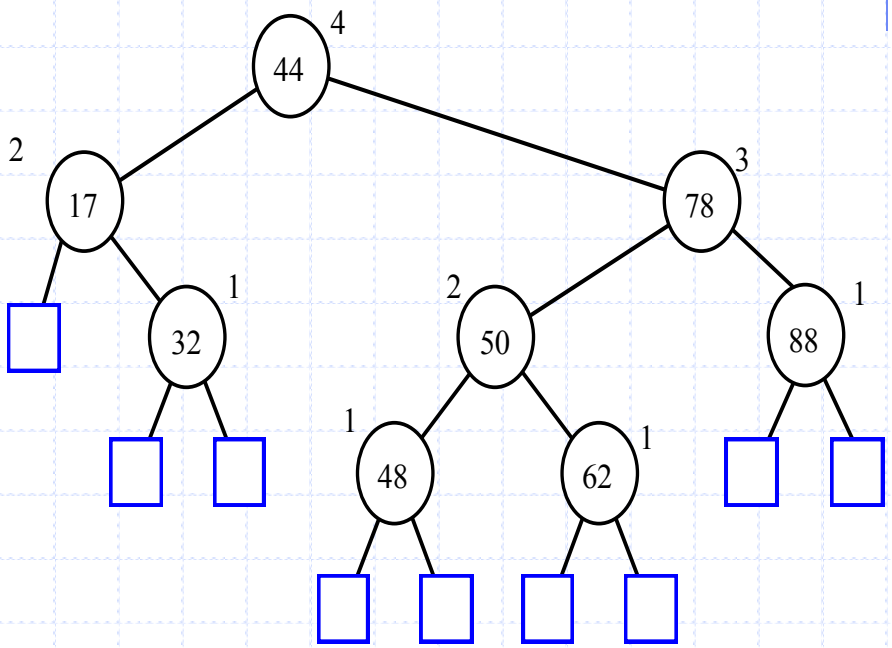
Alberi AVL



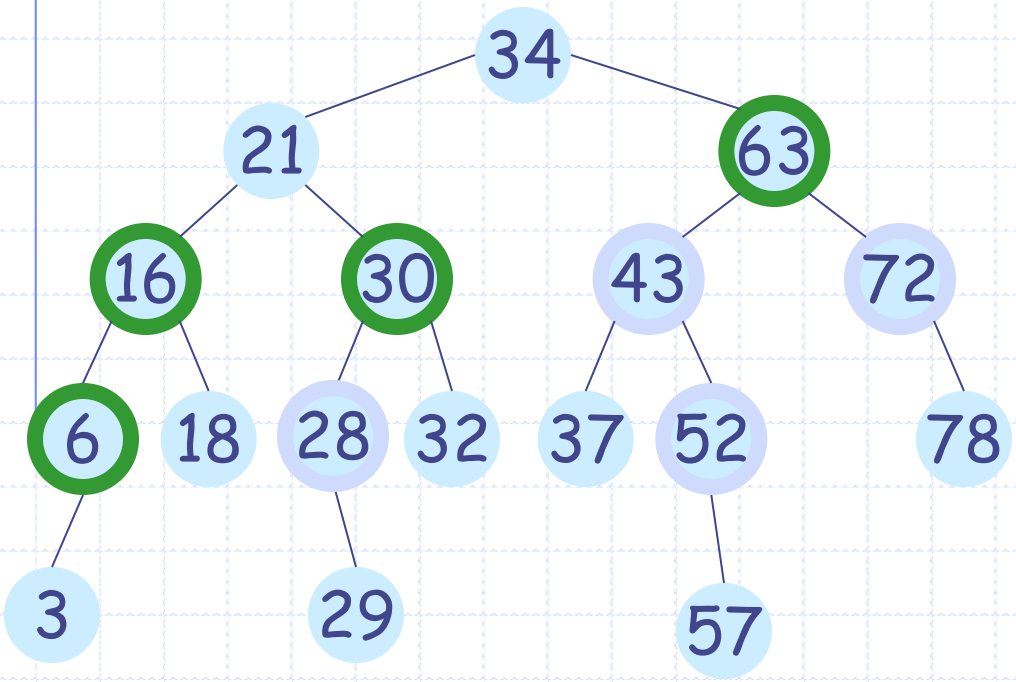
Definizione di albero AVL (§ 10.2)

- **Gli alberi AVL sono bilanciati.**
- Un albero AVL è un ***albero binario di ricerca*** T tale che, per ciascun nodo interno di T, le *altezze dei sottoalberi figli di v possono differire al più di 1.*

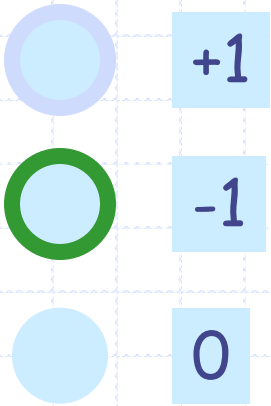


Esempio di albero AVL dove per ogni nodo interno è mostrata l'altezza del suo sottoalbero

fattore di bilanciamento

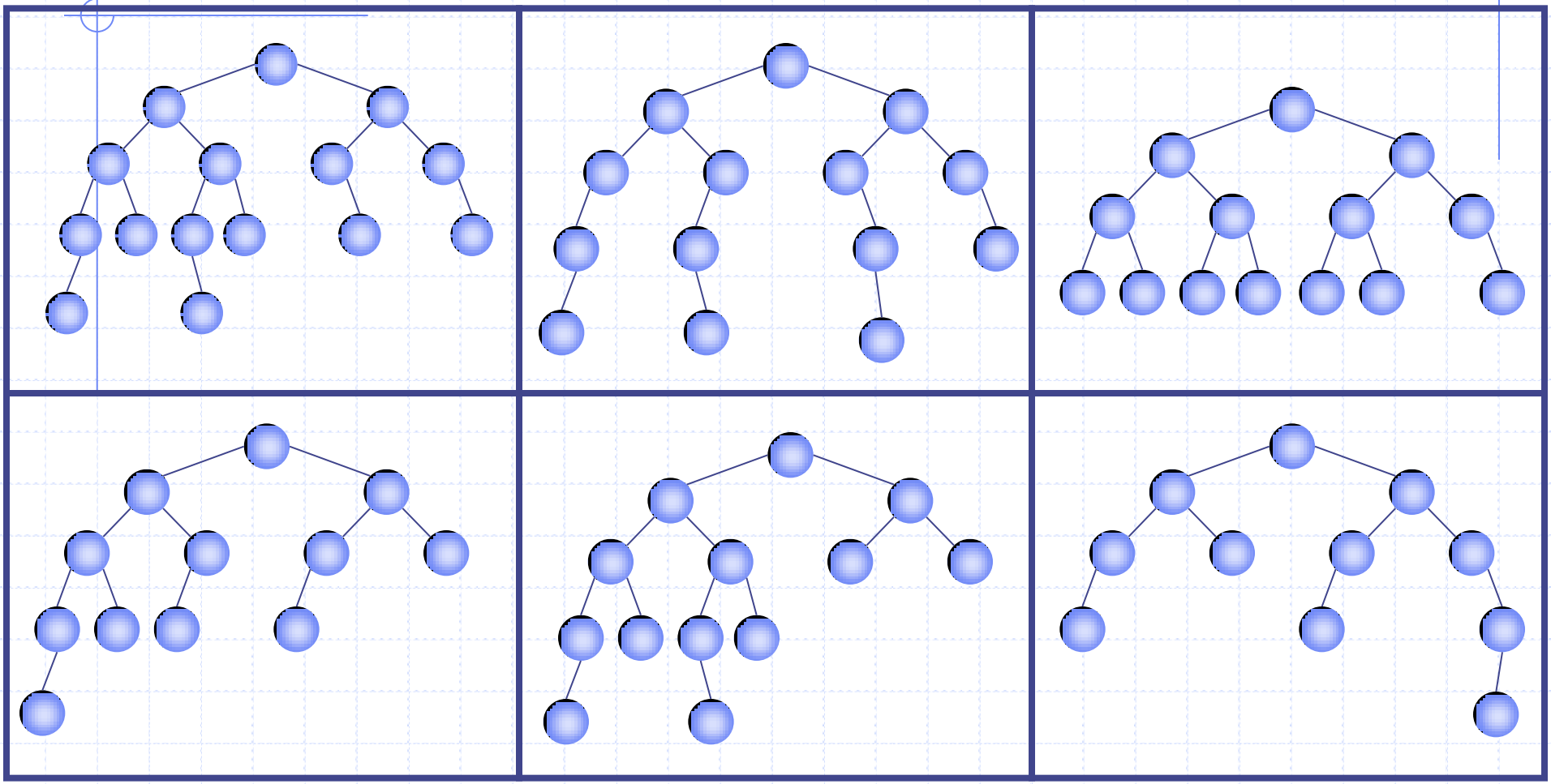


fattore di bilanciamento (FDB):
altezza sottoalbero dx - altezza sottoalbero sx



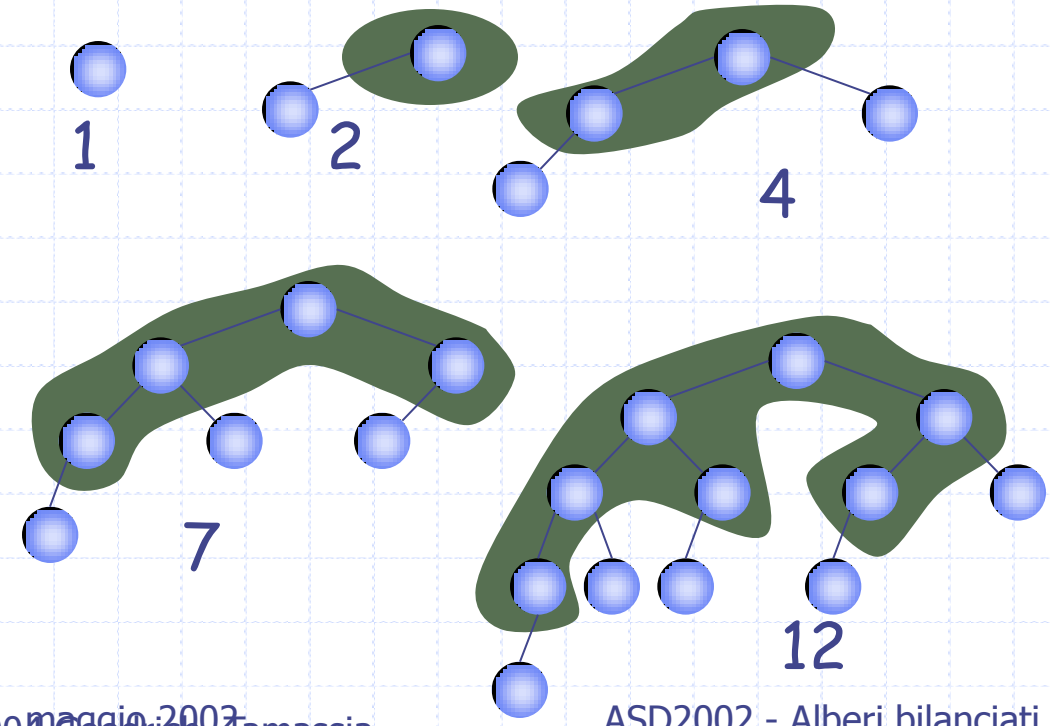
in un albero bilanciato in altezza
$|FDB| \leq 1$, per ogni nodo

alberi AVL?



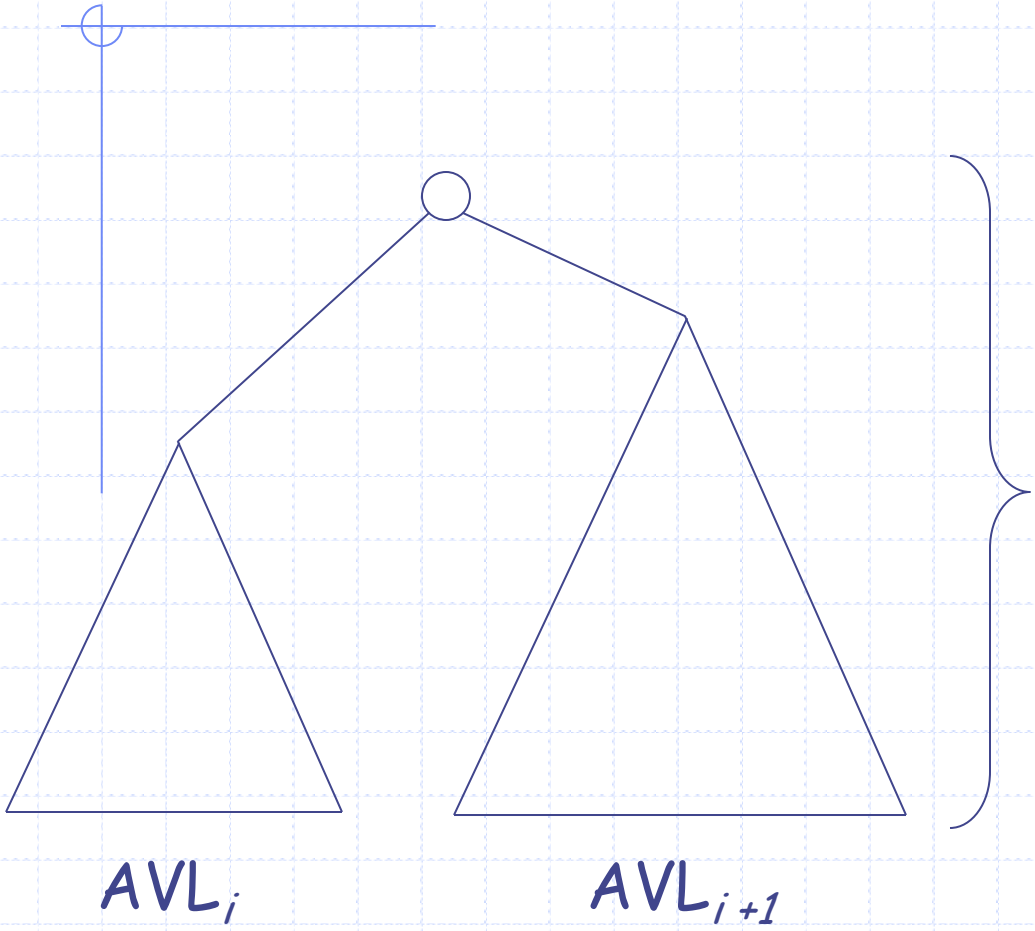
alberi di Fibonacci

- alberi AVL col minimo numero di nodi (fissata l'altezza)



h	F_h	AVL_h
0	0	0
1	1	1
2	1	2
3	2	4
4	3	7
5	5	12
6	8	20
7	13	33

alberi di Fibonacci/2



alberi di Fibonacci
alberi bilanciati di altezza *i* col minimo numero di nodi

Relazioni

$AVL_{i+2} = AVL_i + AVL_{i+1} + 1$

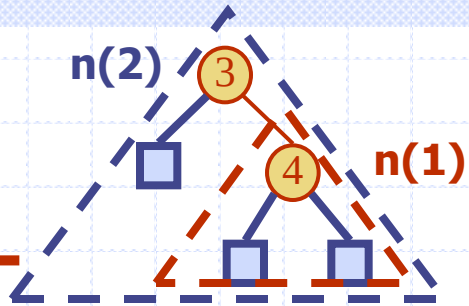
$F_{i+2} = F_i + F_{i+1}$

$AVL_i = F_{i+2} - 1$

alberi di Fibonacci/3

- un albero di Fibonacci ha tutti i fattori di bilanciamento dei nodi interni pari a $\pm$ 1
  - è l'albero bilanciato più vicino alla condizione di non bilanciamento
- un albero di Fibonacci con n nodi ha altezza $< 1.44 \lg(n+2) - 0.328$
  - dimostrato da Adel'son-Vel'skii & Landis
  - $\Rightarrow$ un AVL di n nodi ha altezza $\Theta(\lg n)$

Altezza di un albero AVL

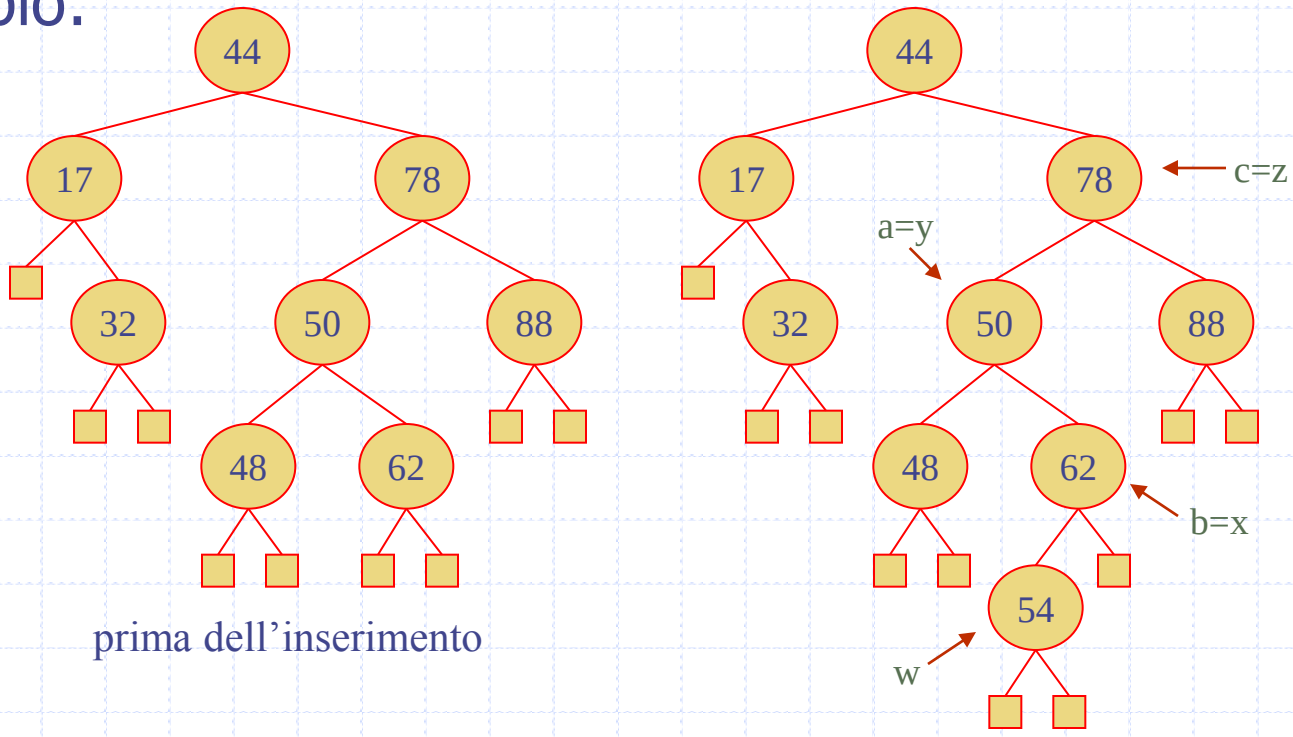


- **Proposizione 10.2**: L'***altezza*** di un albero AVL contenente n chiavi è O(log n).
- **Dimostrazione**: Maggioriamo **n(h)**, il numero minimo di nodi interni di un AVL di altezza h.
- Ovviamente n(1) = 1 ed n(2) = 2
- Per n > 2, un albero AVL di altezza h contiene la radice e due suoi sottoalberi (AVL) di altezza al più n-1, ma uno dei due può avere altezza n-2.
- Cioè: n(h) = 1 + n(h-1) + n(h-2)
- Poiché n(h-1) > n(h-2), ne segue che n(h) > 2n(h-2). Ne segue che

  n(h) > 2n(h-2), n(h) > 4n(h-4), n(h) > 8n(h-6), ... (per induzione),
  n(h) > 2^in(h-2i)
- Risolvendo il caso base otteniamo n(h) > $2^{h/2-1}$
- Passando ai logaritmi: h < 2log n(h) +2 ≤ 2log n +2 (avendo indicato con n il numero effettivo di nodi)
- Ne segue che l'altezza di un albero AVL di n nodi è O(log n)

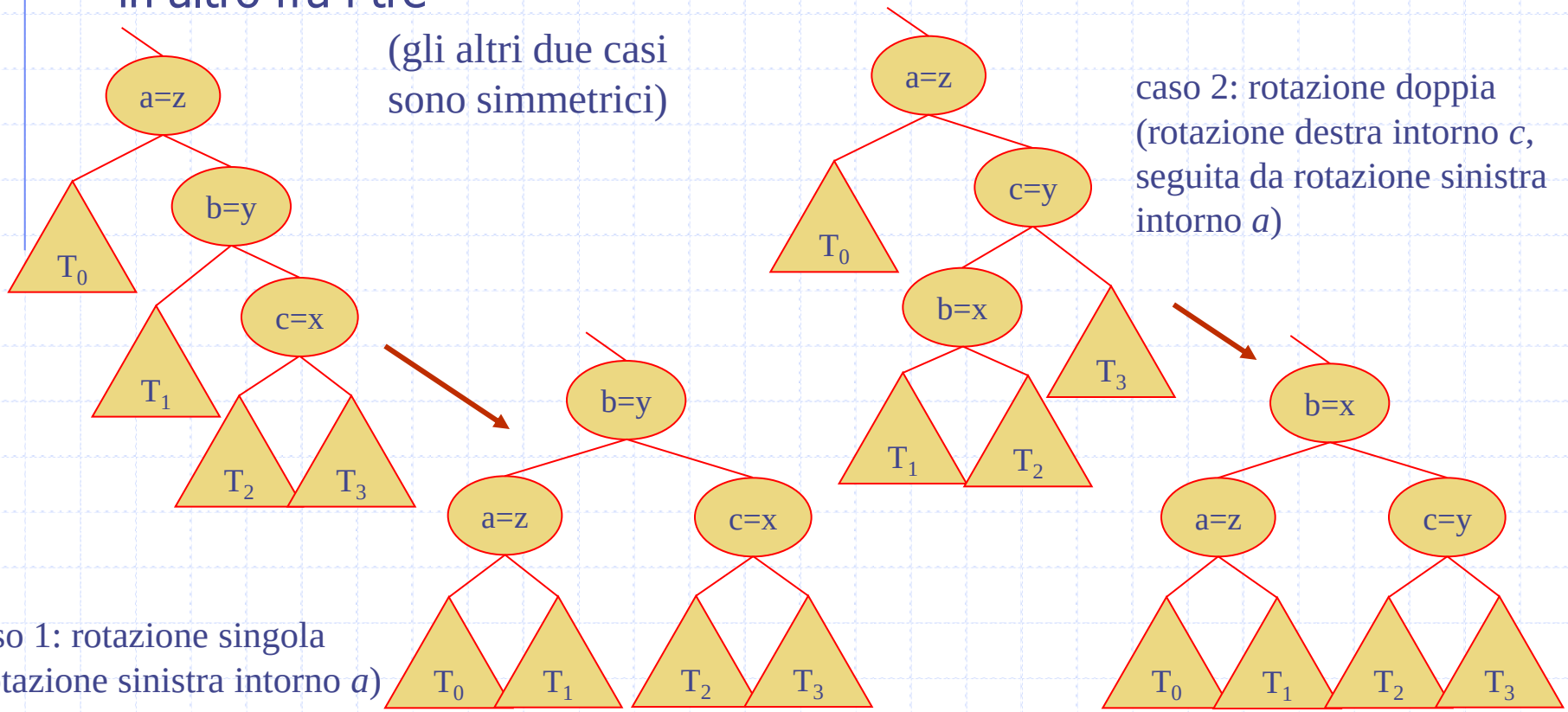
Inserimento in un albero AVL

- L'inserimento è dapprima come in un normale albero binario di ricerca
- Prevede sempre l'espansione di un nodo esterno.
- Esempio:

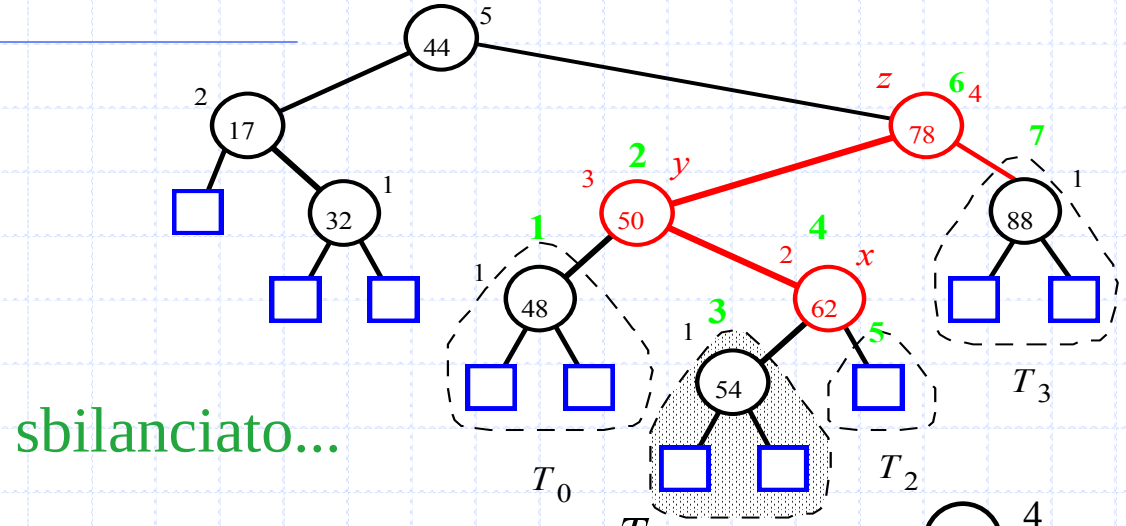


Ristrutturazione della terna di nodi

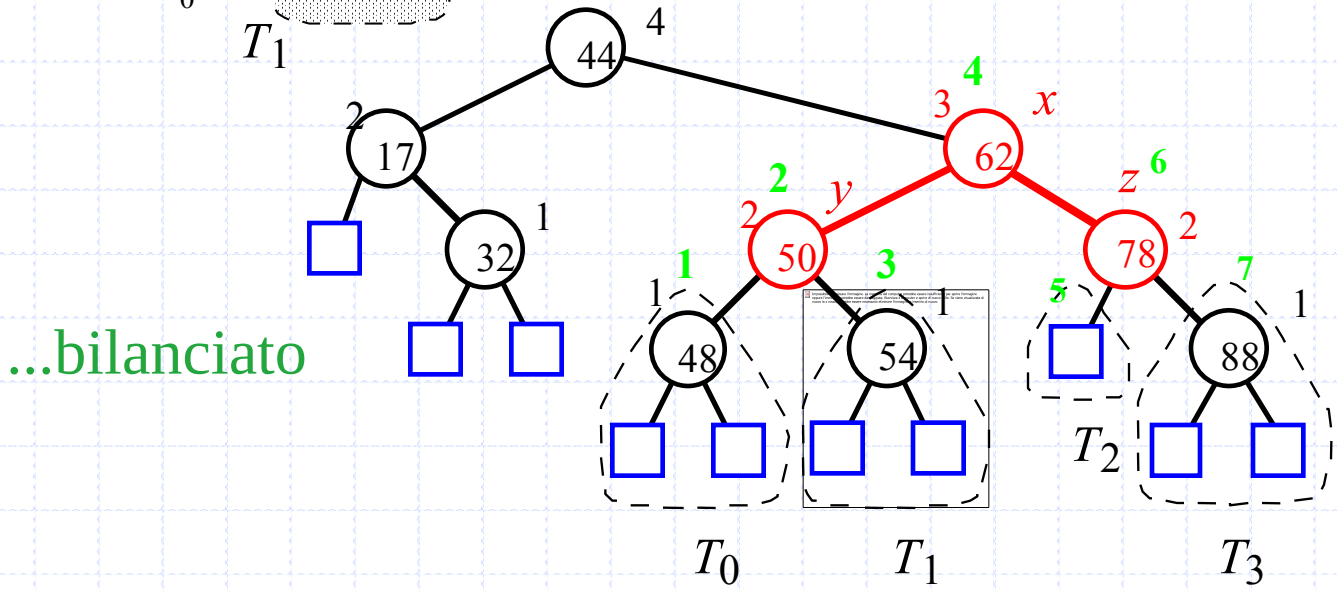
- sia (*a*,*b*,*c*) l'elenco in-ordine di *x*, *y*, *z*
- si eseguono le rotazioni necessarie affinché *b* diventi il nodo più in altro fra i tre



Esempio di inserimento (contin.)



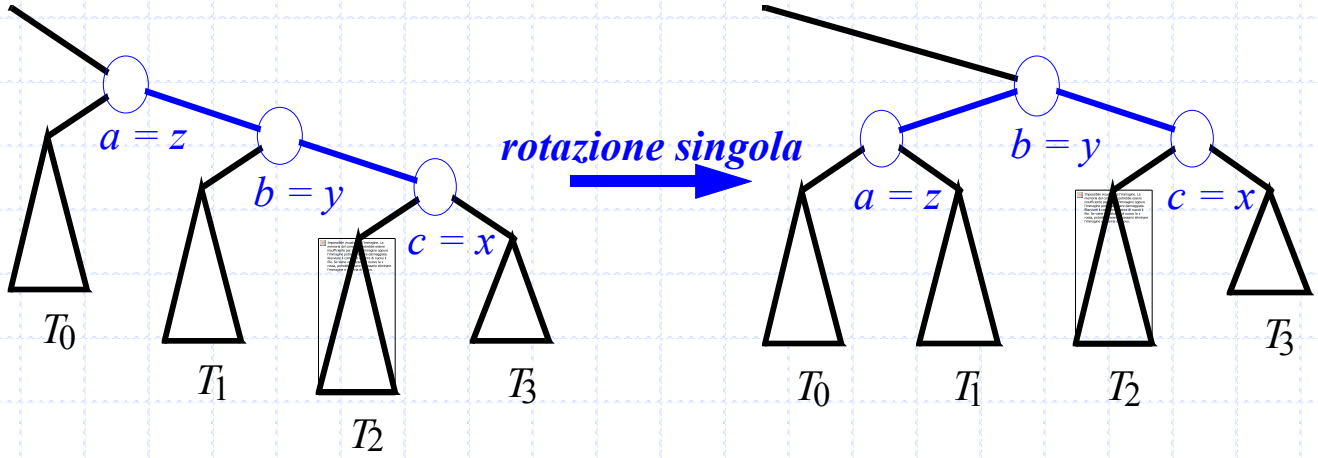
sbilanciato...

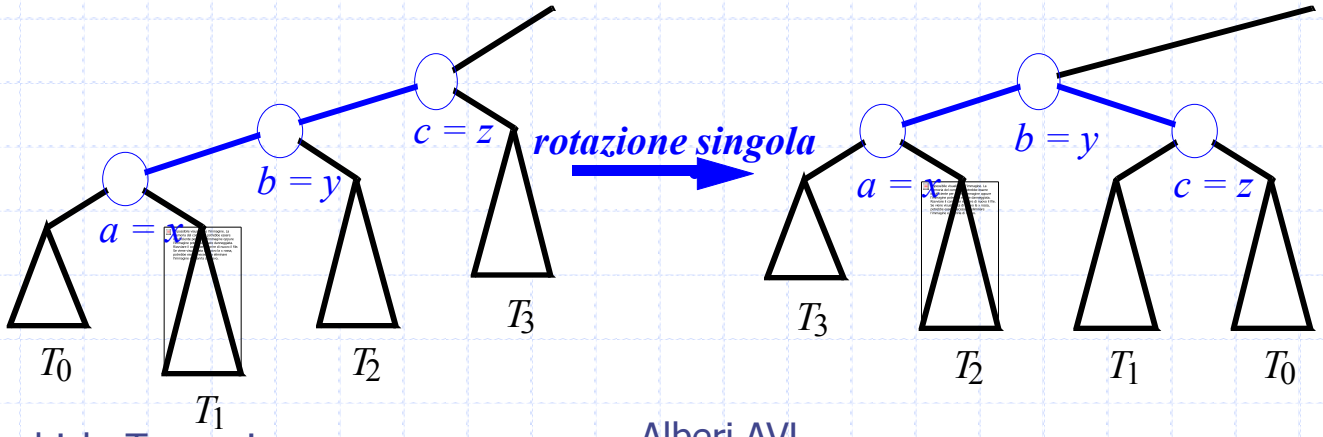


...bilanciato

Ristrutturazione (tramite rotazioni singole)

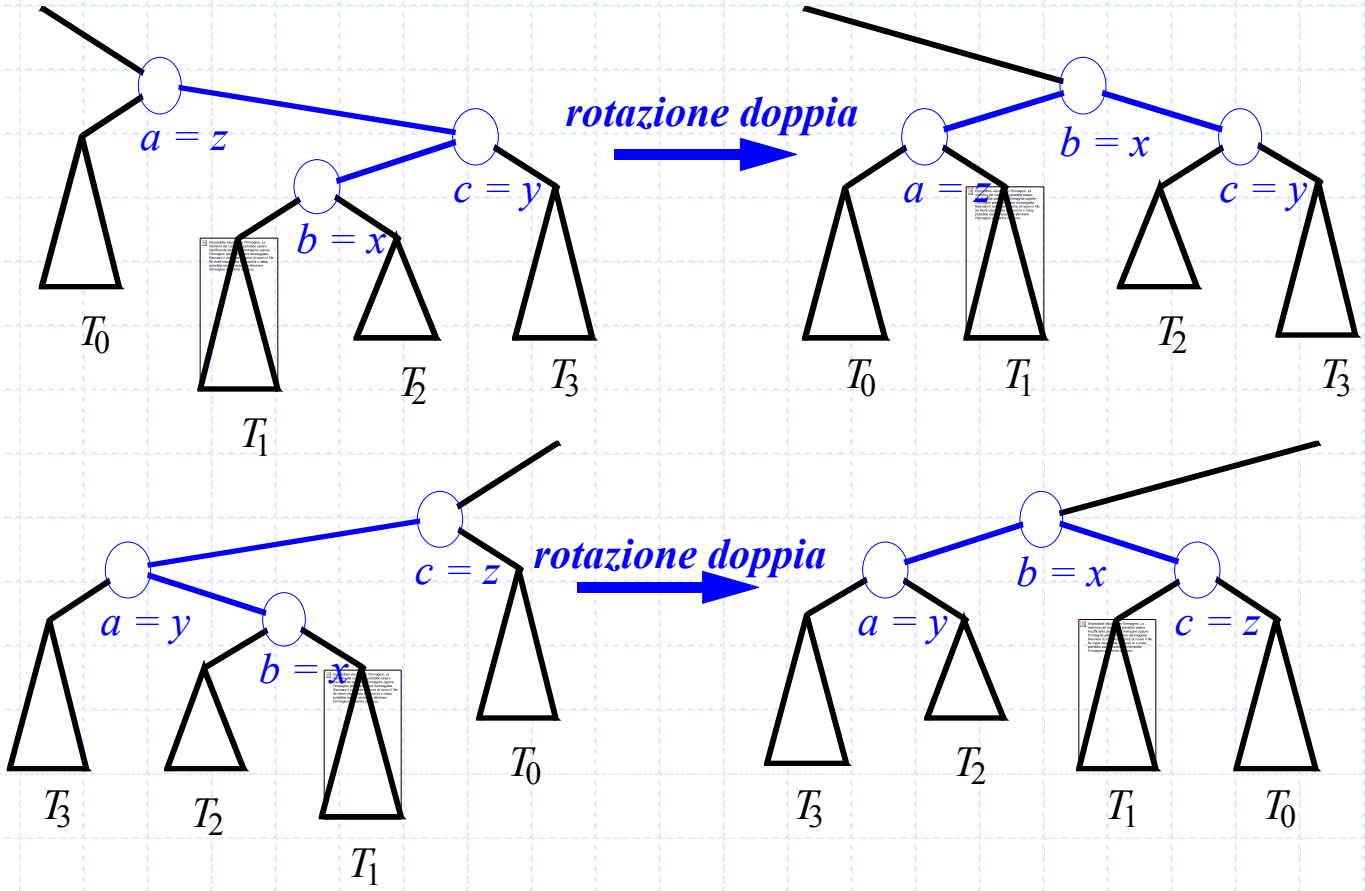
- Rotazioni singole:





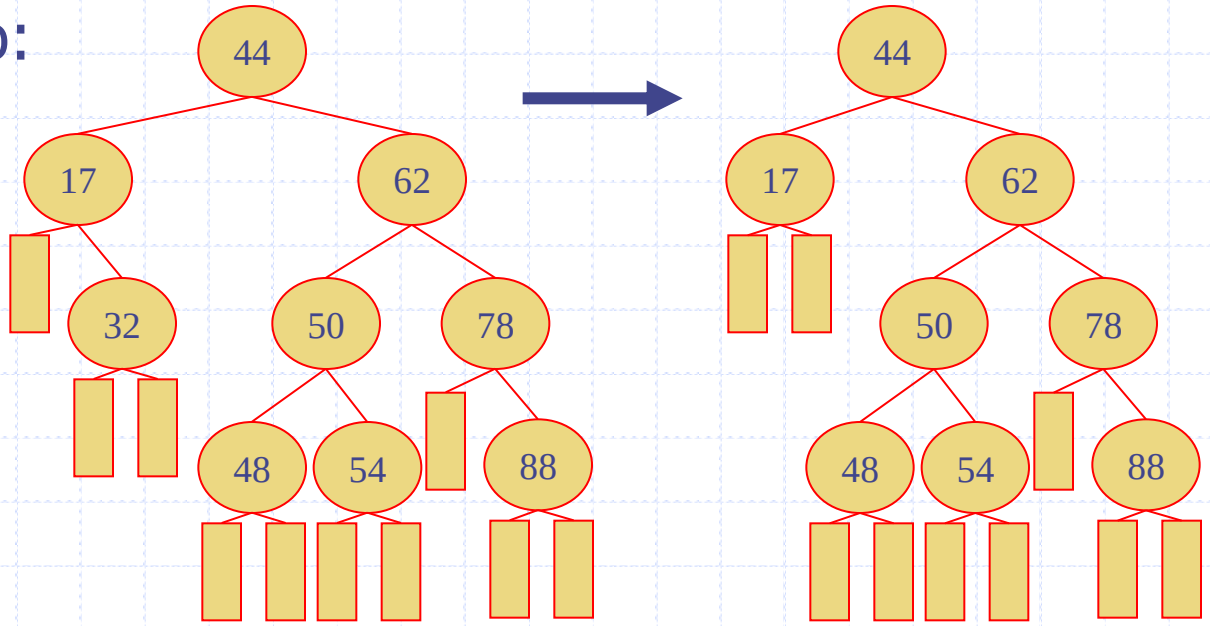
Ristrutturazione (tramite rotazioni doppie)

- Rotazioni doppie:



Cancellazioni negli alberi AVL

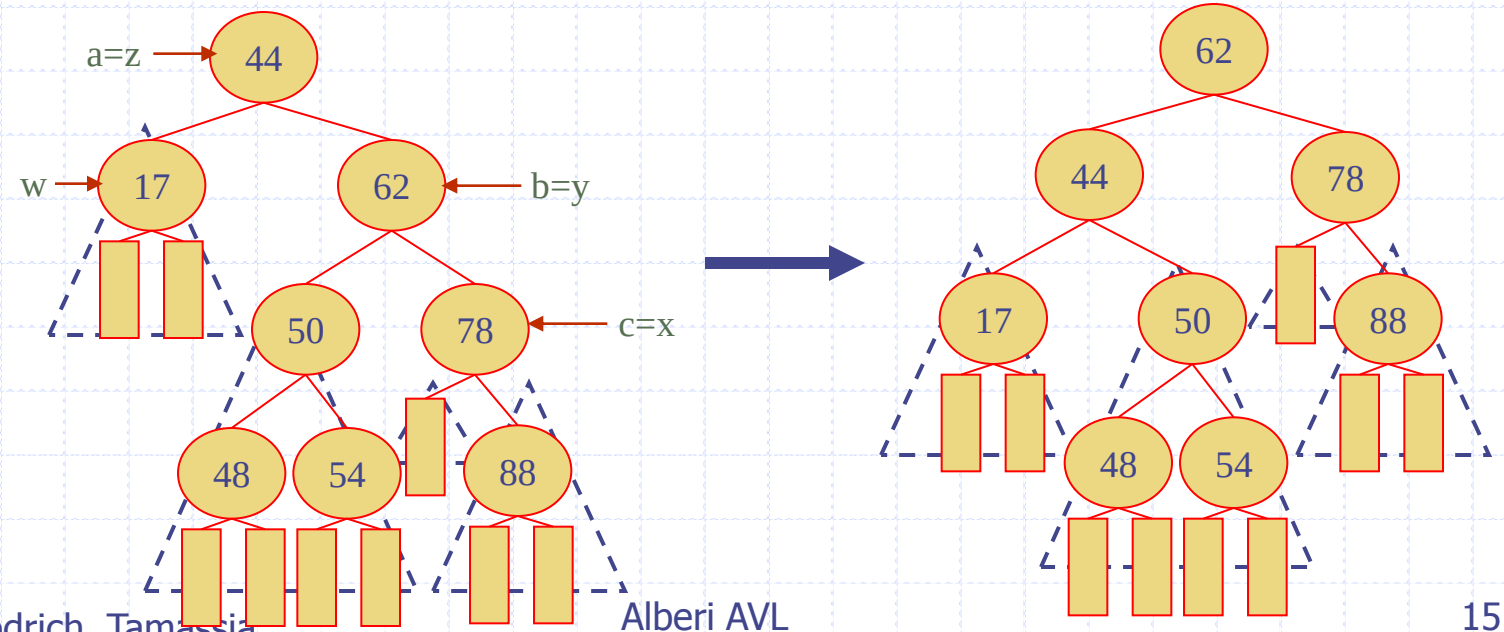
- Nella rimozione si opera dapprima come in un normale albero binario di ricerca, il che significa che il nodo rimosso diverrà un nodo esterno vuoto. Il suo genitore w può tuttavia causare sbilanciamenti.
- Esempio:



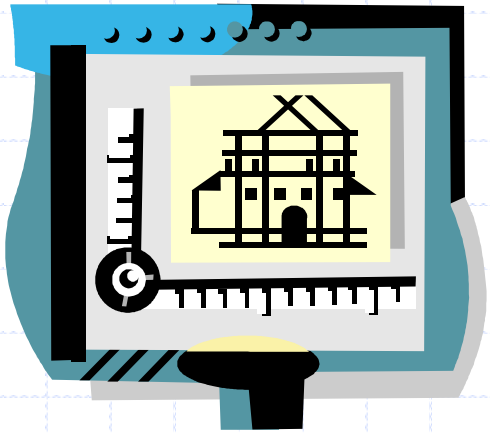
prima della cancellazione di 32 dopo la cancellazione

Re-bilanciamento dopo rimozione

- Sia *z* il primo nodo sbilanciato incontrato durante la navigazione nell'albero da w vero l'alto. Sia y il figlio di z con il più alto sottoalbero (radicato in y) e sia x il figlio di y con il più alto sottoalbero (radicato in x).
- Si esegue restructure(x) per ripristinare il bilanciamento di z.
- Poiché la ristrutturazione può determinare lo sbilanciamento di un antenato di z, è necessario continuare a verificare lo stato del bilanciamento su tutti gli antenati, fino alla radice di T.



Alberi AVL: tempi di esecuzione



- una singola ristrutturazione richiede tempo O(1)
  - usando un albero binario basato su struttura collegata
- find viene eseguito in O(log n)
  - l'altezza dell'albero è O(log n) e non sono necessarie ristrutturazioni
- insert viene eseguito in O(log n)
  - la ricerca iniziale viene fatta in tempo O(log n)
  - una ristrutturazione è sempre sufficiente (tempo O(1))
- remove viene eseguito in O(log n)
  - la ricerca iniziale viene fatta in tempo O(log n)
  - al max un numero logaritmico di ristrutturazioni $O(\log n) \cdot O(1) = O(\log n)$