

ALBERI AVL

PIETRO DI LENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
UNIVERSITÀ DI BOLOGNA

ALGORITMI E STRUTTURE DI DATI
ANNO ACCADEMICO 2021/2022



INTRODUZIONE

- Abbiamo visto che, data una chiave k , con un BST è possibile inserire, rimuovere e cercare nodi in tempo $O(h)$, dove h è l'altezza dell'albero
 - Un albero binario con n nodi ha altezza $h = \Omega(\log n)$ e $h = O(n)$
 - Problema: scrivere un algoritmo che presi in input n chiavi costruisca un BST con altezza $h = \Theta(n)$
 - Problema: scrivere un algoritmo che presi in input n chiavi costruisca un BST con altezza $h = \Theta(\log n)$
- Inserimenti e rimozioni possono **sbilanciare** l'albero
 - Un albero sbilanciato ha al peggio altezza lineare sul numero di nodi
- Obiettivo: **mantenere bilanciato un BST**, anche a seguito di inserimenti e rimozioni di nodi, in modo da avere operazioni di inserimento, rimozione e ricerca con **costo pessimo logaritmico**

- Un **albero AVL** è un albero binario (quasi) perfettamente bilanciato
 - Il nome deriva dagli autori **Adelson-Velsky** e **Landis** (1962)
- Un albero AVL con n nodi supporta le operazioni di base SEARCH, INSERT, DELETE con **costo $O(\log n)$ nel caso pessimo**
- In un albero AVL le operazioni di inserimento e rimozione sono implementate in modo da essere **auto-bilancianti**
 - L'albero viene automaticamente ribilanciato in seguito ad inserimenti e rimozioni che causino sbilanciamento

Fattore di bilanciamento

Il **fattore di bilanciamento** $\beta(v)$ di un nodo v è dato dalla differenza tra l'altezza del sottoalbero sinistro e destro di v :

$$\beta(v) = \text{altezza}(v.\text{left}) - \text{altezza}(v.\text{right})$$

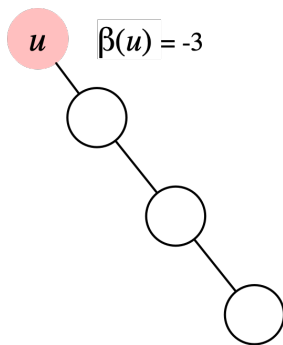
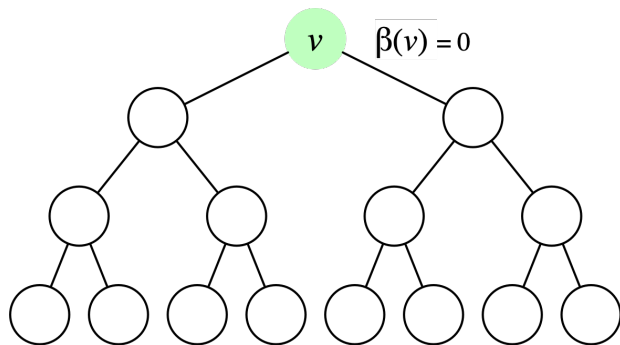
Bilanciamento in altezza

Un albero si dice **bilanciato in altezza** se per ogni nodo v le altezze dei suoi sottoalberi sinistro ($v.\text{left}$) e destro ($v.\text{right}$) differiscono al più di uno

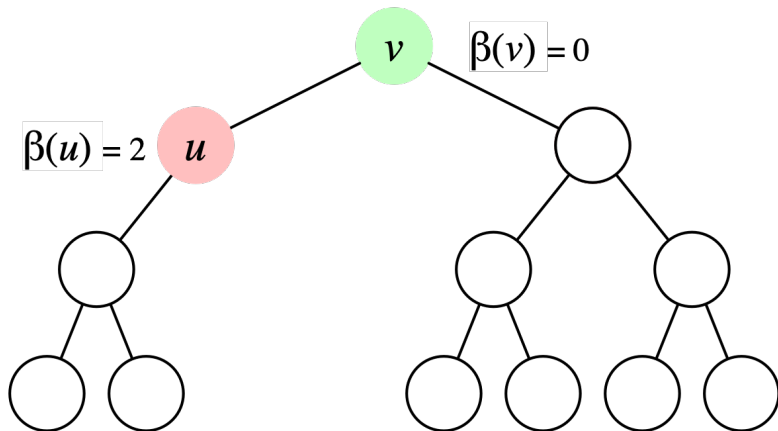
$$|\beta(v)| \leq 1$$

- Un **albero AVL** è un **albero binario bilanciato in altezza**

ESEMPIO: FATTORE DI BILANCIAMENTO

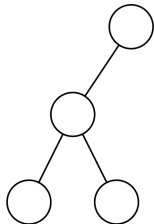


ESEMPIO: ALBERO SBILANCIATO

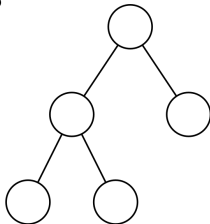


ALBERI AVL?

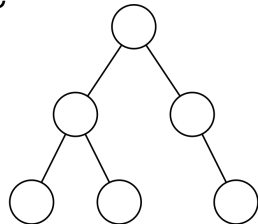
A



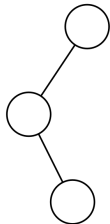
B



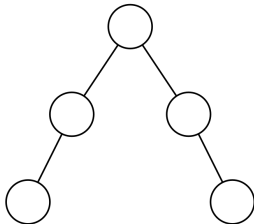
C



D



E

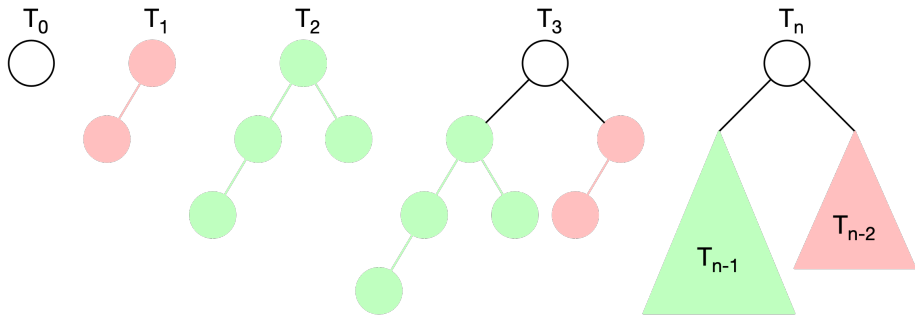


F



ALTEZZA DI UN ALBERO AVL

- Qual è l'altezza massima di un albero AVL?
- Valutiamo l'altezza dell'albero **bilanciato** con **sbilanciamento massimo**
- **Albero di Fibonacci**: per ogni nodo u non foglia $|\beta(u)| = 1$



N.B. Leggermente **diverso** dall'albero della ricorsione di Fibonacci

ALTEZZA DI UN ALBERO DI FIBONACCI

- Sia n_h il numero di nodi di un albero di Fibonacci di altezza h
- Per costruzione abbiamo che

$$n_h = n_{h-1} + n_{h-2} + 1$$

Teorema

Il numero di nodi di un albero di Fibonacci di altezza h è uguale a

$$n_h = F_{h+3} - 1$$

dove F_n è l'ennesimo numero di Fibonacci

- Ricordiamo che $F_1 = F_2 = 1$

n_0	n_1	n_2	n_3
$2 - 1 = 1$	$3 - 1 = 2$	$5 - 1 = 4$	$8 - 1 = 7$

ALTEZZA DI UN ALBERO DI FIBONACCI

■ Dimostriamo per induzione che $n_h = F_{h+3} - 1$

■ **Casi base:**

■ $n_0 = F_3 - 1 = 2 - 1 = 1 \Rightarrow$ vero

■ $n_1 = F_4 - 1 = 3 - 1 = 2 \Rightarrow$ vero

■ **Caso induttivo:** assumiamo vere

$$n_{h-1} = F_{h+2} - 1 \text{ e } n_{h-2} = F_{h+1} - 1$$

dobbiamo dimostrare che $n_h = F_{h+3} - 1$

$$\begin{aligned} n_h &= n_{h-1} + n_{h-2} + 1 \\ &= (F_{h+2} - 1) + (F_{h+1} - 1) + 1 \\ &= F_{h+3} - 1 \Rightarrow \text{vero} \end{aligned}$$

ALTEZZA DI UN ALBERO AVL

- Ricapitolando: un albero di Fibonacci di altezza h ha

$$n_h = F_{h+3} - 1 \text{ nodi}$$

- Ricordiamo che

$$F_h = \Theta(\phi^h) \text{ dove } \phi \approx 1.618$$

da cui otteniamo che

$$n_h = F_{h+3} - 1 = \Theta(\phi^{h+3}) = \Theta(\phi^h)$$

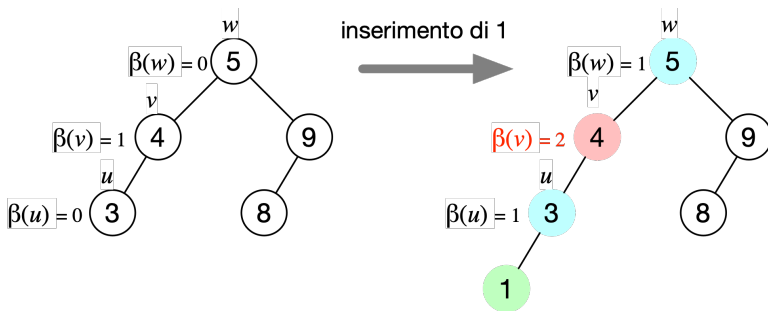
e possiamo quindi concludere che

$$h = \Theta(\log n_h)$$

- Poiché l'albero di Fibonacci con n nodi è l'albero con altezza massima tra tutti gli alberi binari bilanciati con n nodi possiamo concludere che l'altezza di un albero AVL con n nodi è $\Theta(\log n)$

MANTENERE IL BILANCIAMENTO

- La ricerca su un albero AVL viene effettuata come su un BST
- Inserimenti e rimozioni invece richiedono di essere modificati per mantenere il bilanciamento dell'albero



- Per poter verificare il bilanciamento ogni nodo u deve mantenere informazioni sull'altezza del proprio sottoalbero $u.height$
 - Serve per poter calcolare il fattore di bilanciamento β

AGGIORNAMENTO ALTEZZA

```
1: function UPDATE-HEIGHT(AVL  $T$ )
2:   if  $T \neq \text{NIL}$  then
3:      $nh = lh = rh = 0$ 
4:     if  $T.\text{left} \neq \text{NIL}$  then
5:        $lh = T.\text{left}.\text{height}$ 
6:     if  $T.\text{right} \neq \text{NIL}$  then
7:        $rh = T.\text{right}.\text{height}$ 
8:     if  $lh \neq 0$  or  $rh \neq 0$  then
9:        $nh = \max(lh, rh) + 1$ 
10:     $T.\text{height} = nh$ 
```

■ Entrambe costano $O(1)$

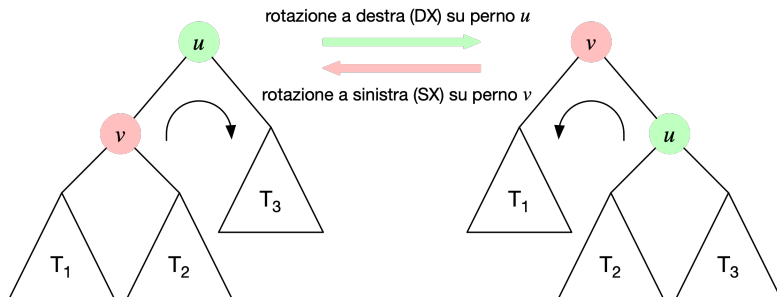
■ UPDATE-HEIGHT nel caso peggiore deve essere richiamata per tutti i nodi in un percorso radice-foglia

■ Possiamo interrompere l'aggiornamento se la nuova altezza non varia rispetto alla precedente

```
1: function  $\beta(\text{AVT } T) \rightarrow \text{INT}$ 
2:    $lh = rh = 0$ 
3:   if  $T.\text{left} \neq \text{NIL}$  then
4:      $lh = T.\text{left}.\text{height}$ 
5:   if  $T.\text{right} \neq \text{NIL}$  then
6:      $rh = T.\text{right}.\text{height}$ 
7:   return  $lh - rh$ 
```

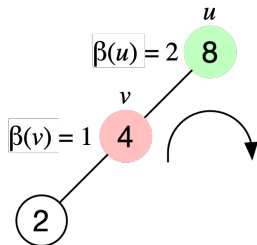
ROTAZIONI

- **Rotazione semplice:** operazione fondamentale per ribilanciare l'albero

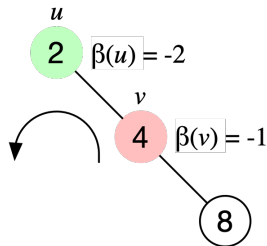
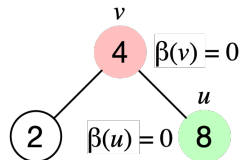


- Preserva la proprietà di ordine dei BST
 - La chiave $u.key \geq v.key$ e maggiore o uguale di ogni chiave in T_1, T_2
 - La chiave $v.key \leq u.key$ e minore o uguale di ogni chiave in T_2, T_3

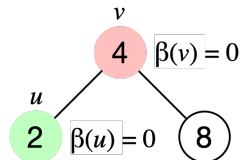
ESEMPI DI ROTAZIONI SEMPLICI



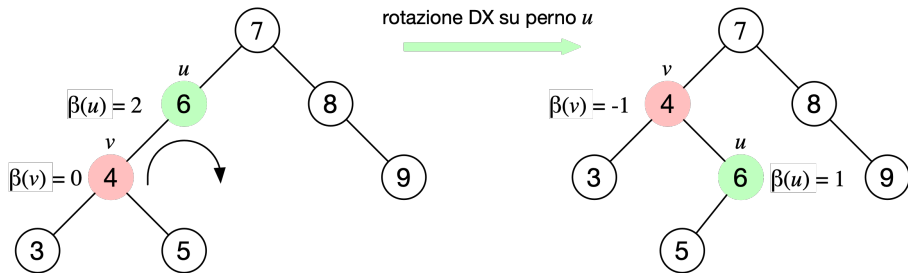
rotazione DX su perno u



rotazione SX su perno u



ESEMPIO DI ROTAZIONE SEMPLICE



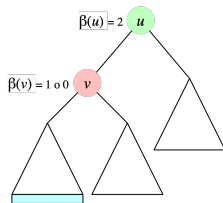
PSEUDOCODE: ROTAZIONE A DESTRA

```
1: function ROTATEDX(AVL  $T$ , AVL  $u$ )  $\rightarrow$  AVL
2:   if  $u \neq \text{NIL}$  and  $u.\text{left} \neq \text{NIL}$  then
3:      $v = u.\text{left}$ 
4:      $v.\text{parent} = u.\text{parent}$ 
5:      $u.\text{parent} = v$ 
6:      $u.\text{left} = v.\text{right}$ 
7:      $v.\text{right} = u$ 
8:     if  $v.\text{parent} == \text{NIL}$  then  $\triangleright v$  is the new root
9:        $T = v$ 
10:    else  $\triangleright$  parent update
11:      if  $v.\text{parent}.\text{left} == u$  then
12:         $v.\text{parent}.\text{left} = v$ 
13:      else
14:         $v.\text{parent}.\text{right} = v$ 
15:    return  $T$ 
```

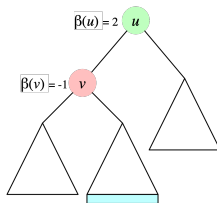
- Costo: $O(1)$
- Ritorna la nuova (eventuale) radice dell'albero AVL
- ROTATESX simmetrica

SBILANCIAMENTI

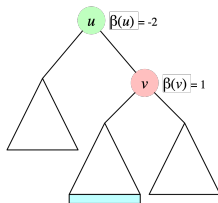
- Assumiamo di avere un albero AVL bilanciato, in cui il sottoalbero radicato in un nodo u diventa sbilanciato in seguito ad una operazione di inserimento o rimozione
- Abbiamo quattro casi (simmetrici a due a due)
 - Sbilanciamento SS: $\beta(u) = 2$ e $\beta(u.left) \geq 0$
 - Sbilanciamento DD: $\beta(u) = -2$ e $\beta(u.right) \leq 0$
 - Sbilanciamento SD: $\beta(u) = 2$ e $\beta(u.left) = -1$
 - Sbilanciamento DS: $\beta(u) = -2$ e $\beta(u.right) = 1$



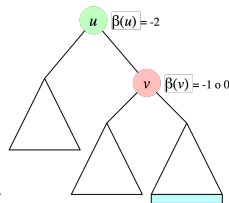
SS (Sinistro-Sinistro)



SD (Sinistro-Destro)

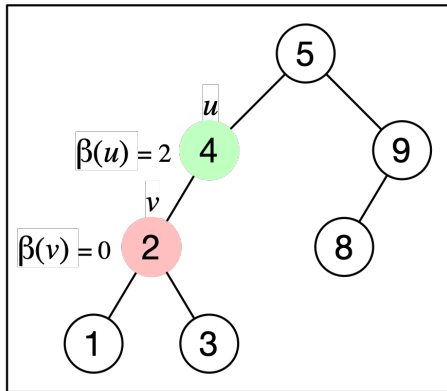
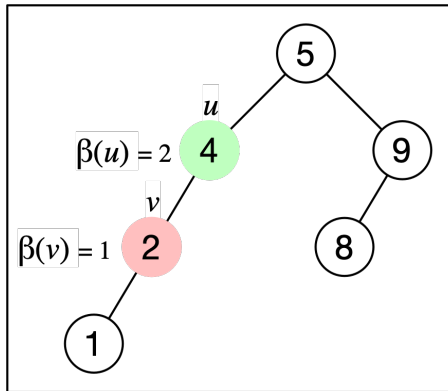


DS (Destro-Sinistro)

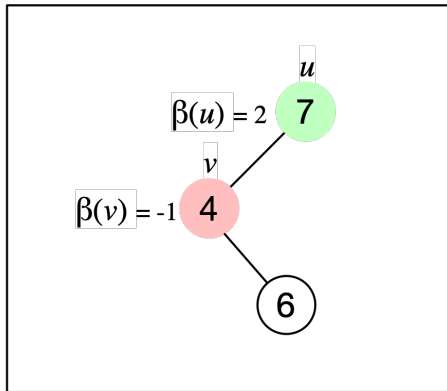
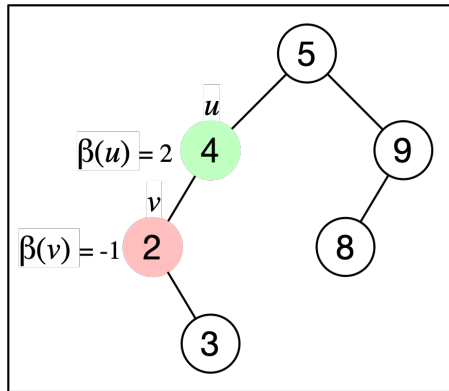


DD (Destro-Destro)

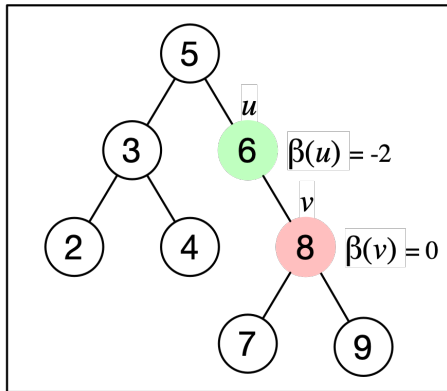
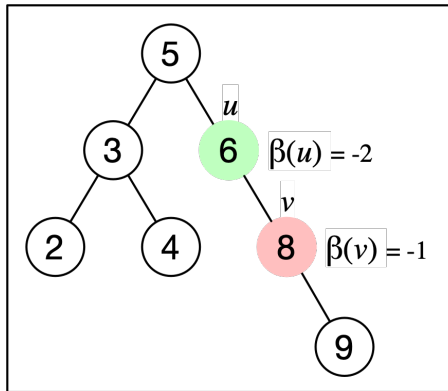
ESEMPI: SBILANCIAMENTI SS



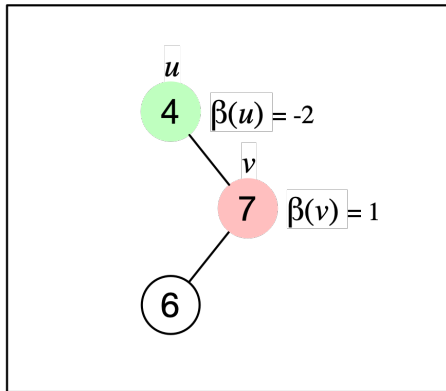
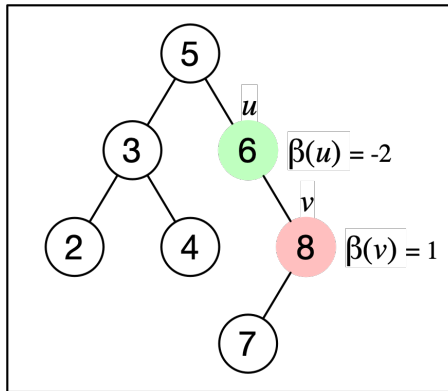
ESEMPI: SBILANCIAMENTI SD



ESEMPI: SBILANCIAMENTI DD



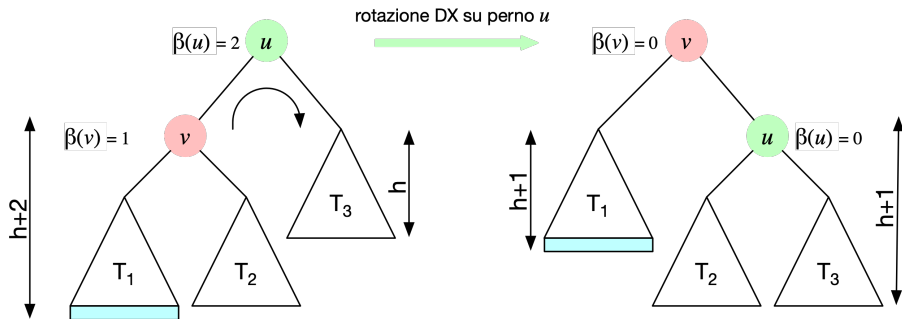
ESEMPI: SBILANCIAMENTI DS



SBILANCIAMENTO SS \Rightarrow ROTAZIONE DX 1/2

- Ribilanciamento per uno sbilanciamento SS \Rightarrow Rotazione DX

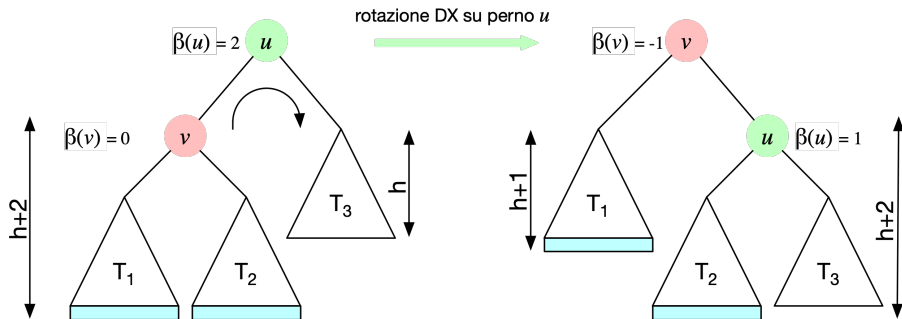
$$\beta(u) = 2 \text{ e } \beta(u.\text{left}) = 1$$



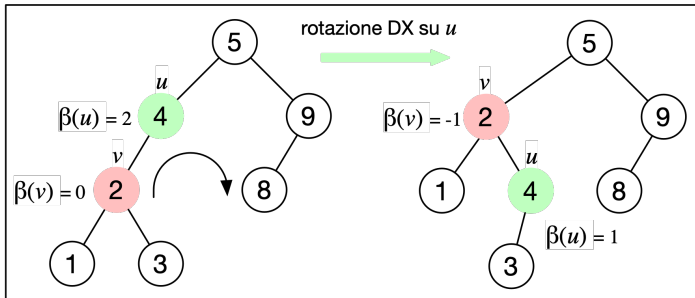
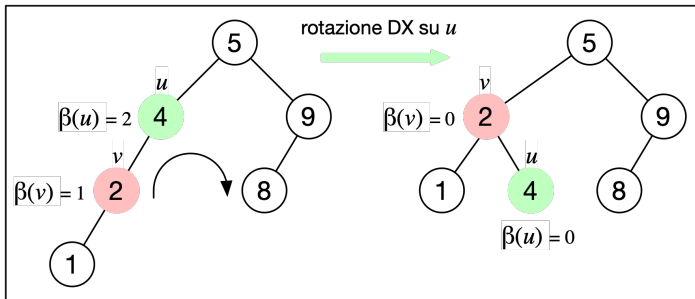
SBILANCIAMENTO SS \Rightarrow ROTAZIONE DX 2/2

- Ribilanciamento per uno sbilanciamento SS \Rightarrow Rotazione DX

$$\beta(u) = 2 \text{ e } \beta(u.\text{left}) = 0$$



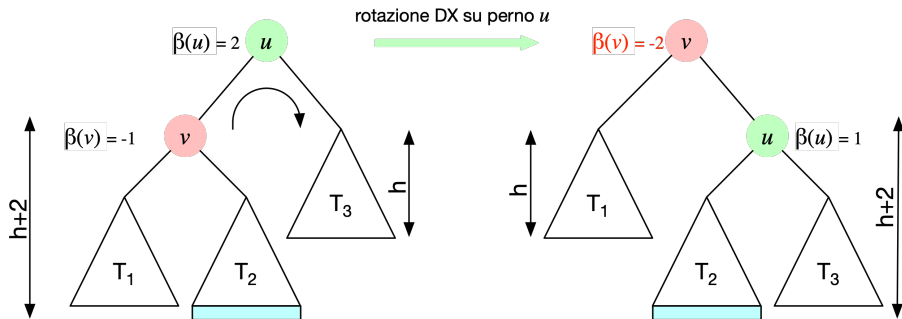
ESEMPI: SBILANCIAMENTO SS \Rightarrow ROTAZIONE DX



SBILANCIAMENTO SD \Rightarrow ROTAZIONE DX?

- Ribilanciamento per uno sbilanciamento SD con Rotazione DX

$$\beta(u) = 2 \text{ e } \beta(u.\text{left}) = -1$$



- Non funziona!!

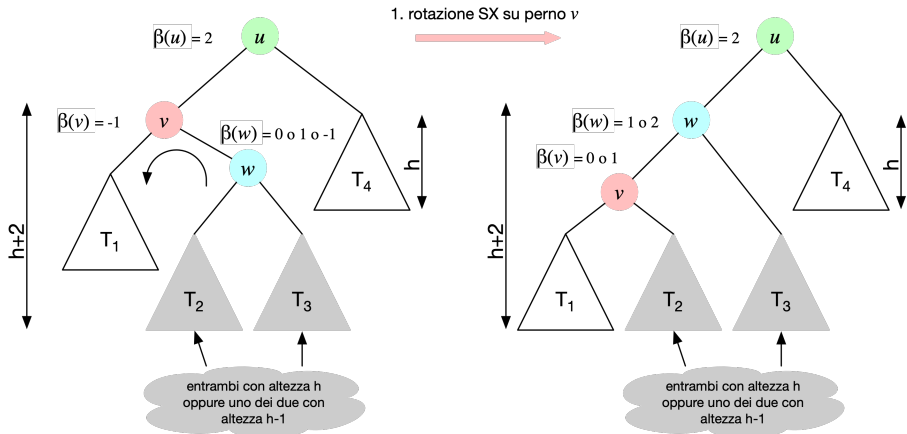
- Abbiamo ottenuto uno sbilanciamento DS

SBILANCIAMENTO SD \Rightarrow ROTAZIONE SXDX 1/2

- Ribilanciamento per uno sbilanciamento SD \Rightarrow Rotazione SXDX

$$\beta(u) = 2 \text{ e } \beta(u.\text{left}) = -1$$

- Step 1: rotazione SX con perno $u.\text{left}$

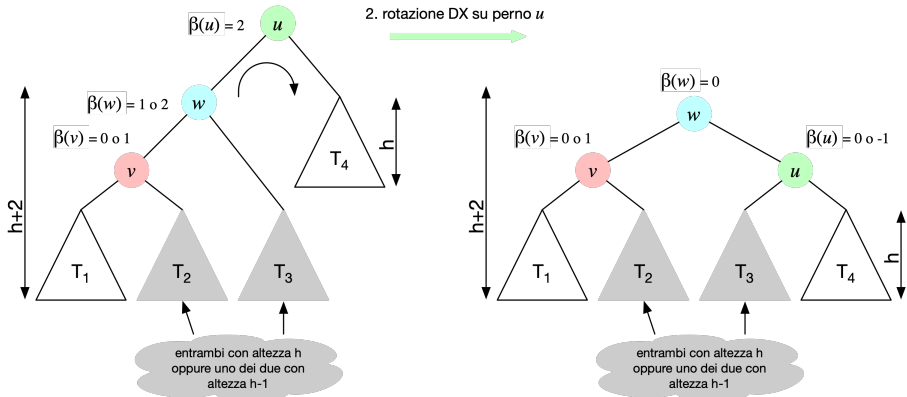


SBILANCIAMENTO SD \Rightarrow ROTAZIONE SXDX 2/2

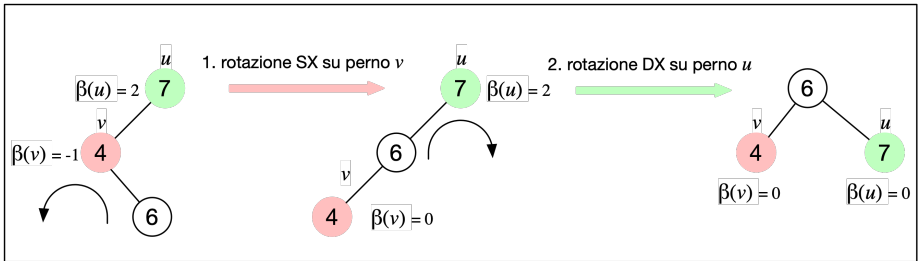
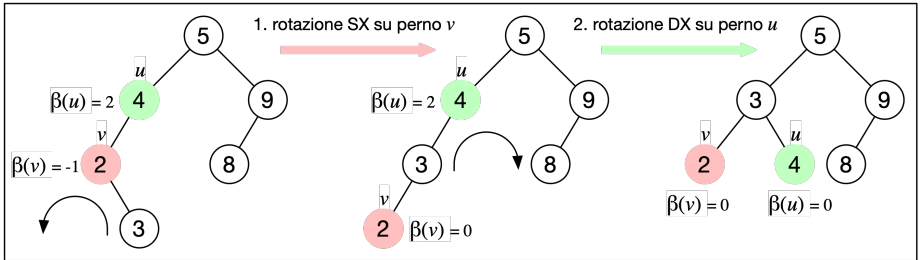
- Ribilanciamento per uno sbilanciamento SD \Rightarrow Rotazione SXDX

$$\beta(u) = 2 \text{ e } \beta(u.\text{left}) = -1$$

- Step 2: rotazione DX con perno u



ESEMPI: SBILANCIAMENTO SD \Rightarrow ROTAZIONE SXDX



RIASSUNTO: SBILANCIAMENTI E ROTAZIONI

- Sbilanciamento SS: $\beta(u) = 2$ e $\beta(u.left) \geq 0$
 - Rotazione DX su u
- Sbilanciamento DD: $\beta(u) = -2$ e $\beta(u.right) \leq 0$
 - Rotazione SX su u
- Sbilanciamento SD: $\beta(u) = 2$ e $\beta(u.left) = -1$
 - Rotazione SX su $u.left$
 - Rotazione DX su u
- Sbilanciamento DS: $\beta(u) = -2$ e $\beta(u.right) = 1$
 - Rotazione DX su $u.right$
 - Rotazione SX su u

PSEUDOCODE: BILANCIAMENTO

```
1: function BALANCE(AVL  $T$ , AVL  $u$ )  $\rightarrow$  AVL
2:   if  $u \neq \text{NIL}$  and  $|\beta(u)| == 2$  then
3:     if  $\beta(u) == 2$  then
4:       if  $\beta(u.\text{left}) == -1$  then  $\triangleright$  Sbilanciamento SD
5:         ROTATESX( $T$ ,  $u.\text{left}$ )
6:        $T = \text{ROTATEDX}(T, u)$ 
7:     else
8:       if  $\beta(u.\text{right}) == 1$  then  $\triangleright$  Sbilanciamento DS
9:         ROTATEDX( $T$ ,  $u.\text{right}$ )
10:       $T = \text{ROTATESX}(T, u)$ 
11:   return  $T$ 
```

■ Costo: $O(1)$

■ Ritorna la (eventualmente nuova) radice dell'albero AVL

Java (asdlab.libreria.AlberiRicerca.AlberoAVL)

Metodo per il ribilanciamento di un albero AVL tramite rotazione

```
private void ruota(Nodo v) {
    switch (bil(v)) {
        case +2:
            if (bil(alb.sin(v)) > -1) // Caso SS (0, +1)
                ruotaDes(v);
            else { // Caso SD (-1)
                ruotaSin(alb.sin(v));
                ruotaDes(v);
            }
            break;
        case -2:
            if (bil(alb.des(v)) < +1) // Caso DD (-1, 0)
                ruotaSin(v);
            else { // Caso DS (+1)
                ruotaDes(alb.des(v));
                ruotaSin(v);
            }
            break;
    }
}
```


INSERIMENTO IN UN ALBERO AVL

- 1 Si inserisce il nuovo nodo come per i BST
 - Costo nel caso pessimo: $O(\log n)$

 - 2 Si riaggiornano le altezze dei sotto-alberi, eventualmente cambiate
 - Nel caso peggiore, tale aggiornamento dovrà essere effettuato per tutti i nodi nel cammino dal nodo inserito fino alla radice
 - Costo nel caso pessimo: $O(\log n)$

 - 3 Se un nodo presenta fattore di bilanciamento ± 2 (nodo critico), occorre ribilanciare l'albero con la procedura di ribilanciamento
 - N.B. In caso di inserimento abbiamo al più un nodo critico
 - Costo nel caso pessimo: $O(1)$
- Costo dell'inserimento in un albero AVL nel caso pessimo: $O(\log n)$

Metodo per l'inserimento in un albero AVL

```
public Rif insert(Object e, Comparable k) {  
    InfoAVL i = new InfoAVL(e, k);  
    Nodo v = super.insert(i);  
    Nodo p = alb.padre(v);  
    while (p != null) {  
        if (bil(p)==-2 || bil(p)==2) break;  
        aggiornaAltezza(p);  
        p = alb.padre(p);  
    }  
    if (p != null) ruota(p);  
    return i;  
}
```

RIMOZIONE IN UN ALBERO AVL

- 1 Si rimuove il nodo come per i BST
 - Costo nel caso pessimo: $O(\log n)$

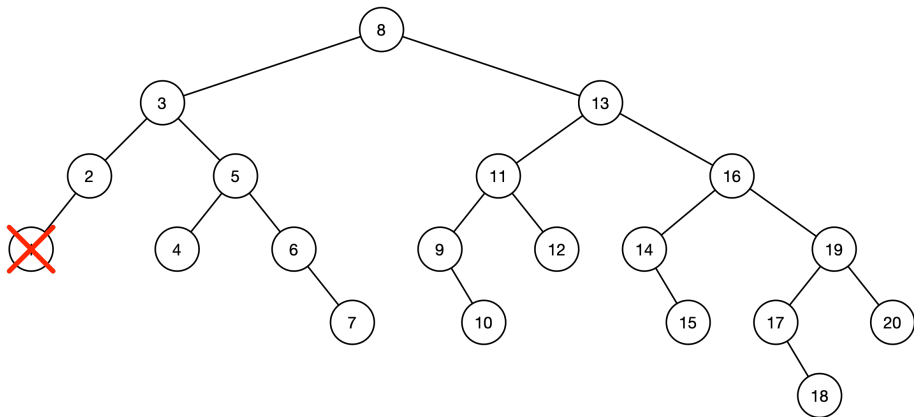
 - 2 Si riaggiornano le altezze dei sotto-alberi, eventualmente cambiate
 - Nel caso peggiore, tale aggiornamento dovrà essere effettuato per tutti i nodi nel cammino dal nodo inserito fino alla radice
 - Costo nel caso pessimo: $O(\log n)$

 - 3 Se un nodo presenta fattore di bilanciamento ± 2 (nodo critico), occorre ribilanciare l'albero con la procedura di ribilanciamento
 - N.B. In caso di rimozione potremmo avere più un nodi critici
 - Tutti i nodi critici sono in un unico percorso radice-foglia
 - Costo nel caso pessimo: $O(\log n)$
-
- Costo della rimozione in un albero AVL nel caso pessimo: $O(\log n)$

Metodo per la rimozione in un albero AVL

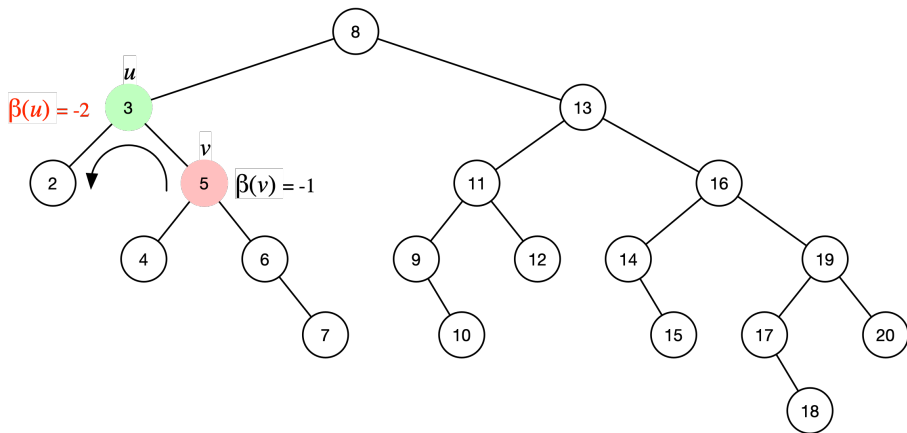
```
public void delete(Rif i) {  
    Nodo p = super.delete((InfoAVL)i);  
    while (p != null) {  
        if (bil(p)==-2 || bil(p)==2) ruota(p);  
        else aggiornaAltezza(p);  
        p = alb.padre(p);  
    }  
}
```

ESEMPIO: ROTAZIONI A CASCATA 1/4



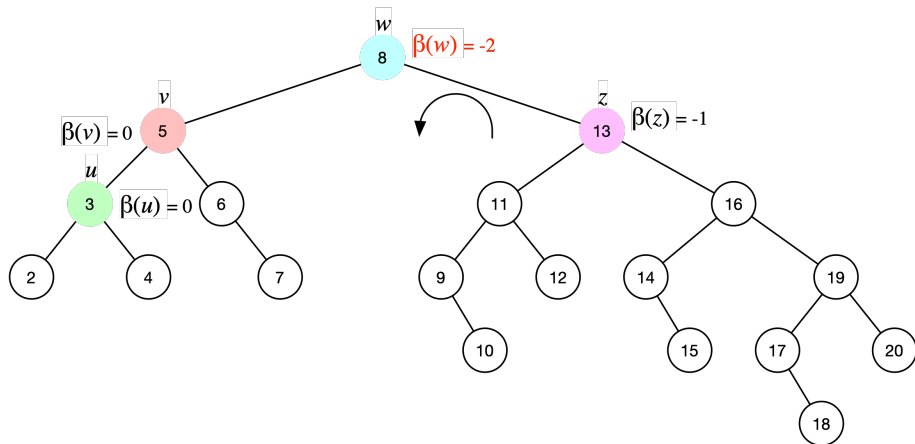
Albero AVL bilanciato prima della rimozione del nodo con chiave 1

ESEMPIO: ROTAZIONI A CASCATA 2/4



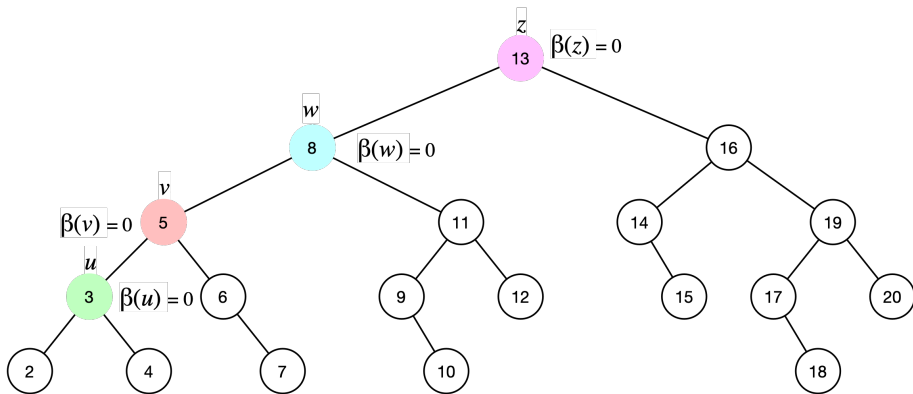
Ribilanciamento: rotazione SX con perno u

ESEMPIO: ROTAZIONI A CASCATA 3/4



Ribilanciamento: rotazione SX con perno w

ESEMPIO: ROTAZIONI A CASCATA 4/4



Albero ribilanciato

	SEARCH	INSERT	DELETE
Array ordinati	$O(\log n)$	$O(n)$	$O(n)$
Liste concatenate	$O(n)$	$O(1)$	$O(n)$
Alberi BST	$O(h)$	$O(h)$	$O(h)$
Alberi AVL	$O(\log n)$	$O(\log n)$	$O(\log n)$

- I costi si riferiscono tutti al caso pessimo
- Nota: $h = O(n)$ è l'altezza dell'albero