Alberi AVL

Pietro Di Lena

Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna

Algoritmi e Strutture di Dati
Anno Accademico 2022/2023

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

Alberi AVL (Adelson-Velsky e Landis, 1962)

- Un albero AVL è un albero binario (quasi) perfettamente bilanciato
  - Il nome deriva dagli autori Adelson-Velsky e Landis
- 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

Definizioni

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:

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

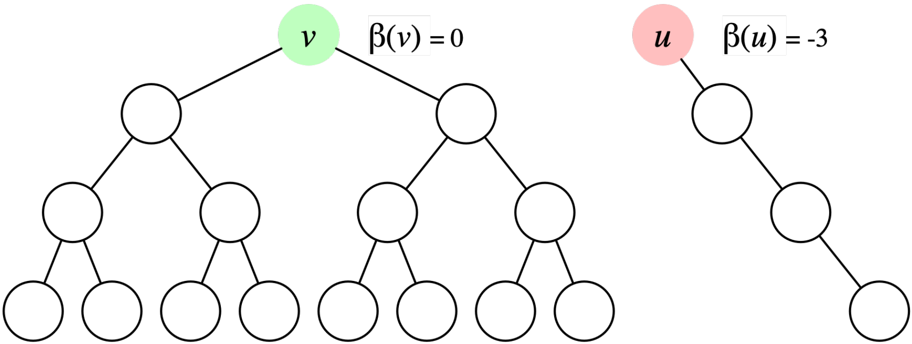
Bilanciamento in altezza

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

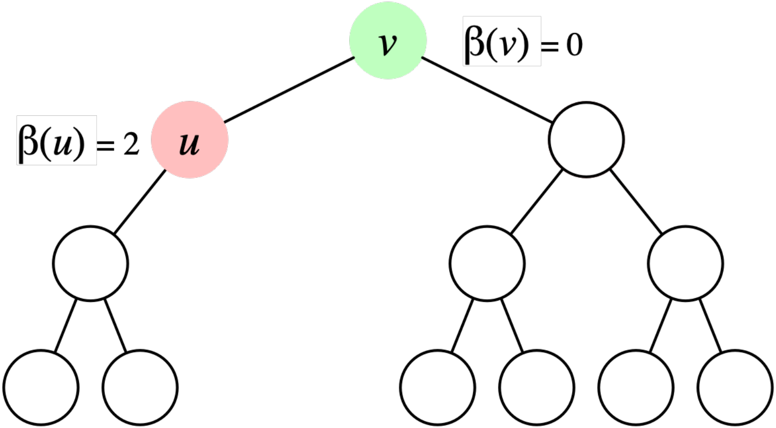
$$|\beta(v)| \leq 1$$

- Un albero AVL è un albero binario bilanciato in altezza

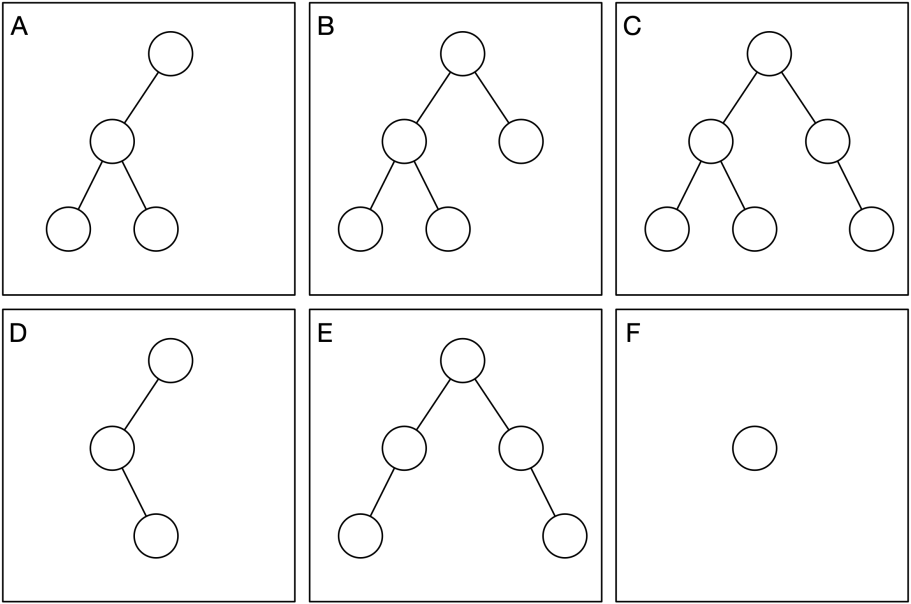
Esempio: fattore di bilanciamento



Esempio: albero sbilanciato

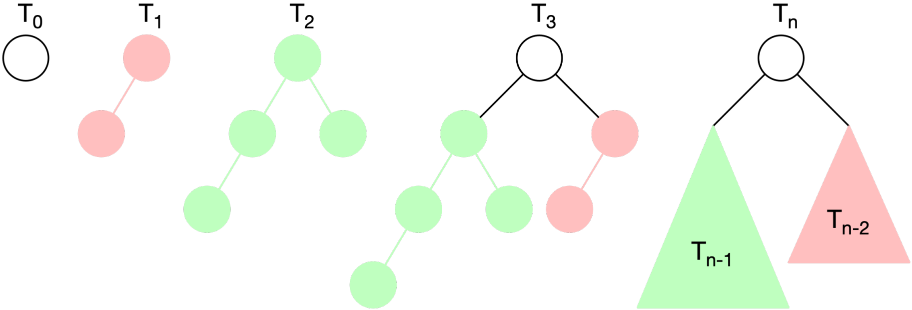


Alberi AVL?



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

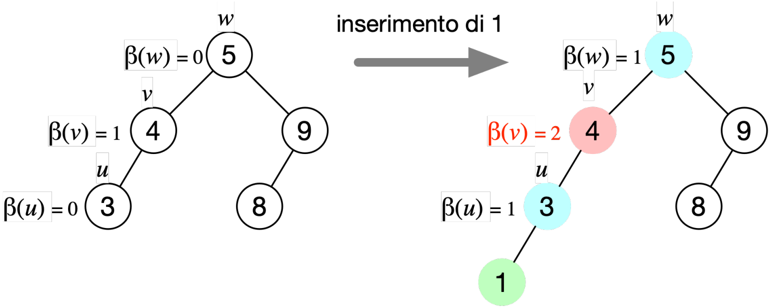
$$\begin{aligned} n_h &= n_{h-1} + n_{h-2} + 1 \\ &= (F_{h+2} - 1) + (F_{h+1} - 1) + 1 \\ &= F_{h+3} - 1 \Rightarrow \text{vero} \end{aligned}$$

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

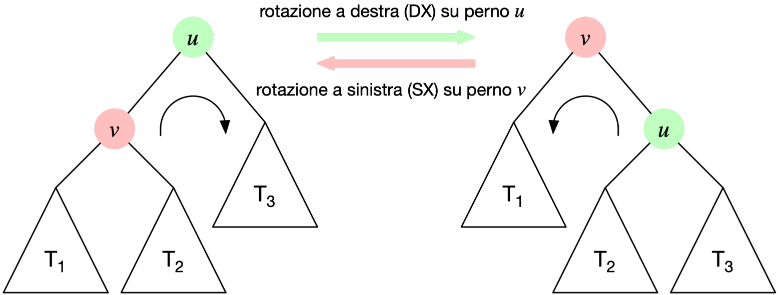
```
function UPDATE-HEIGHT(NODE T)
    if T ≠ NIL then
        nh = lh = rh = 0
        if T.left ≠ NIL then
            lh = T.left.height
        if T.right ≠ NIL then
            rh = T.right.height
        if T.left≠NIL or T.right≠NIL then
            nh = max(lh, rh) + 1
        T.height = nh
```

```
function β(NODE T) → INT
    lh = rh = 0
    if T.left ≠ NIL then
        lh = T.left.height
    if T.right ≠ NIL then
        rh = T.right.height
    return lh − rh
```

- Entrambe costano $O(1)$
- UPDATE-HEIGTH 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

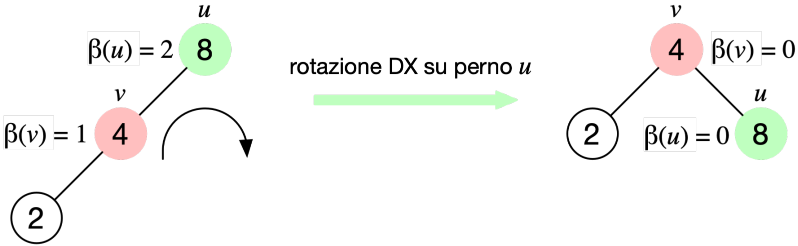
Rotazioni

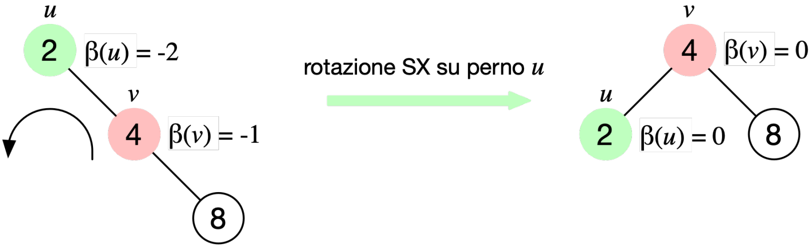
- Rotazione semplice: operazione fondamentale per ribilanciare l'albero



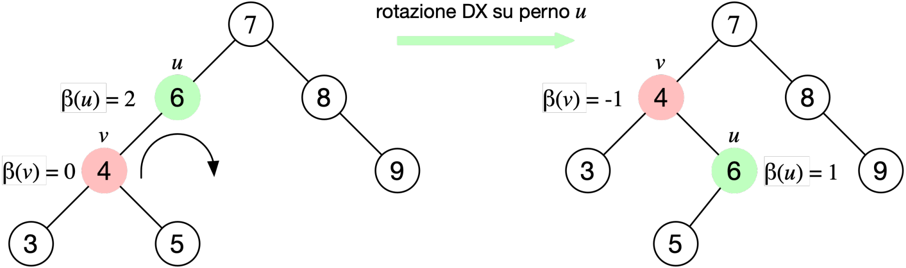
- Preserva la proprietà di ordine dei BST
  - La chiave $u.key \geq v.key$ e maggiore o uguale di ogni chiave in T_1, T_2
  - La chiave $v.key \leq u.key$ e minore o uguale di ogni chiave in T_2, T_3

Esempi di rotazioni semplici





Esempio di rotazione semplice



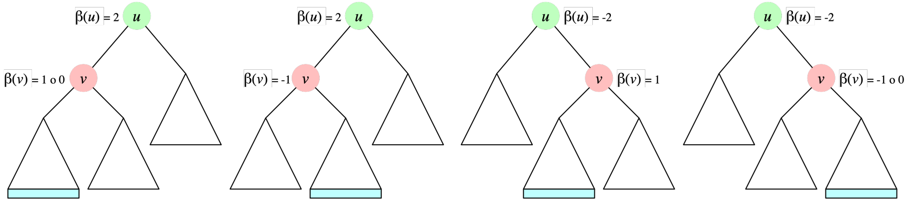
Pseudocodice: rotazione a destra

```
function rotateDX(AVL T, Node u)
    if u ≠ NIL and u.left ≠ NIL then
        v = u.left
        v.parent = u.parent
        u.parent = v
        u.left = v.right
        v.right = u
        if v.parent == NIL then          ▷ v is the new root
            T.root = v
        else                             ▷ parent update
            if v.parent.left == u then
                v.parent.left = v
            else
                v.parent.right = v
```

- Costo: $O(1)$
- rotateSX simmetrica

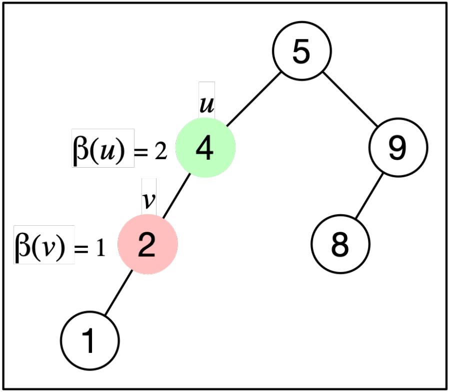
Sbilanciamenti

- 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 $\beta(u.right) = 1$

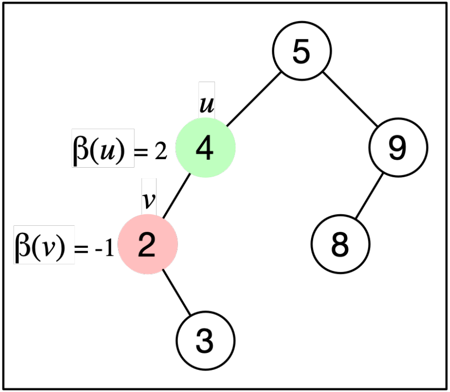


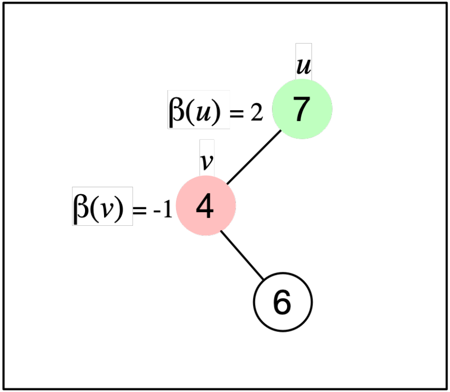
SS (Sinistro-Sinistro) SD (Sinistro-Destro) DS (Destro-Sinistro) DD (Destro-Destro)

Esempi: sbilanciamenti SS

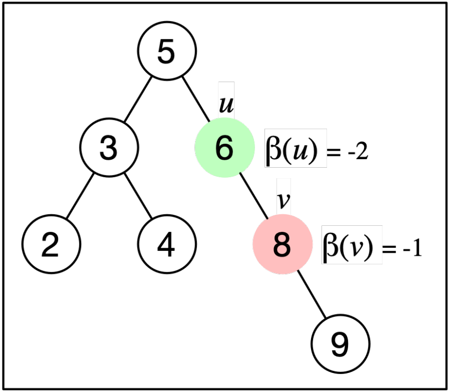


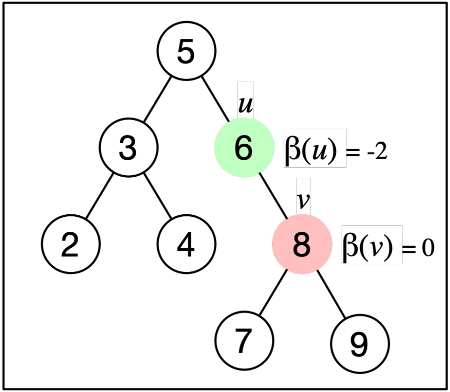
Esempi: sbilanciamenti SD



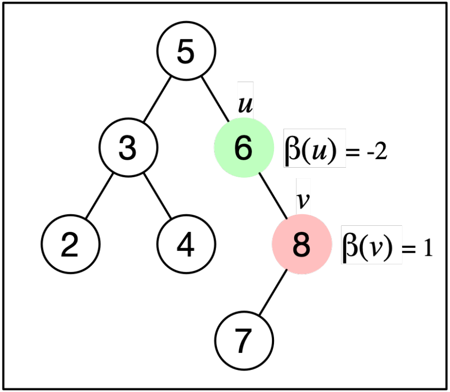


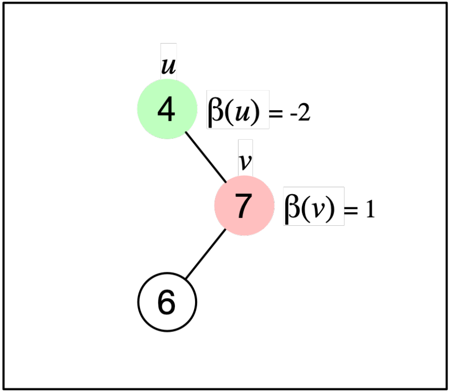
Esempi: sbilanciamenti DD





Esempi: sbilanciamenti DS

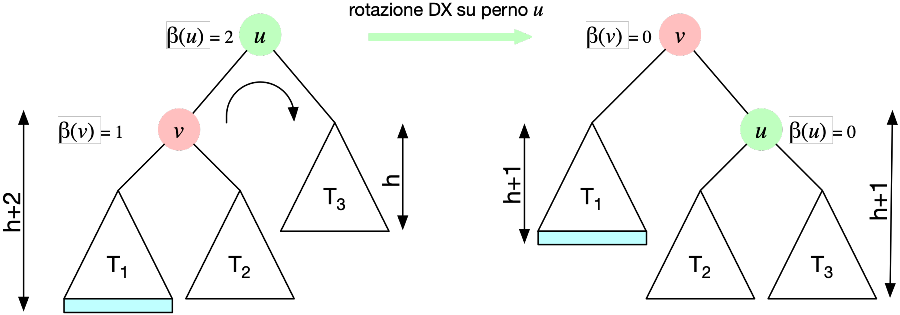




Sbilanciamento SS $\Rightarrow$ Rotazione DX 1/2

- Ribilanciamento per uno sbilanciamento SS $\Rightarrow$ Rotazione DX

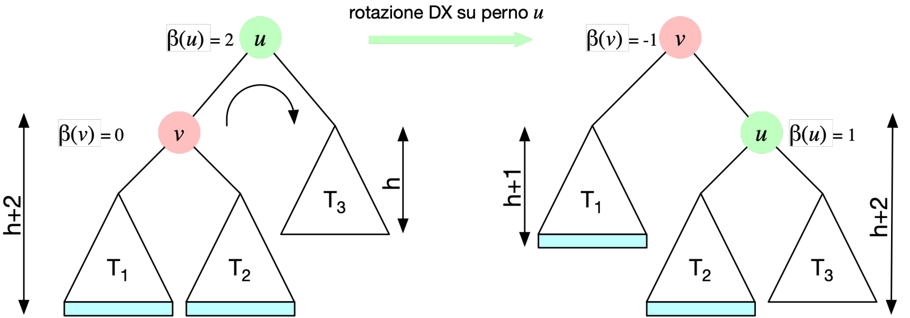
$$\beta(u) = 2 \text{ e } \beta(u.left) = 1$$



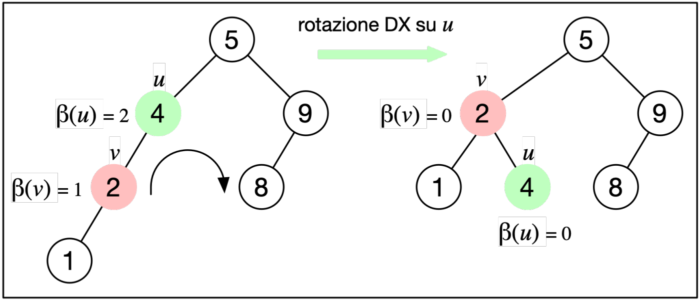
Sbilanciamento SS ⇒ Rotazione DX 2/2

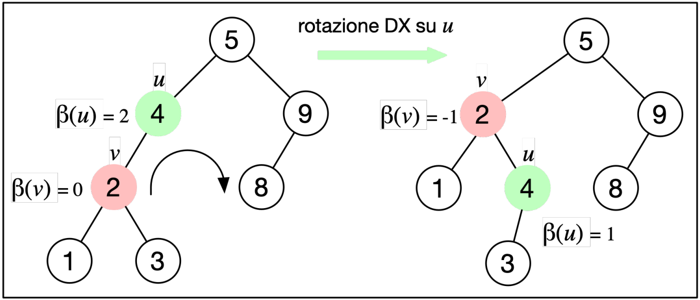
- Ribilanciamento per uno sbilanciamento SS ⇒ Rotazione DX

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



Esempi: sbilanciamento SS ⇒ Rotazione DX

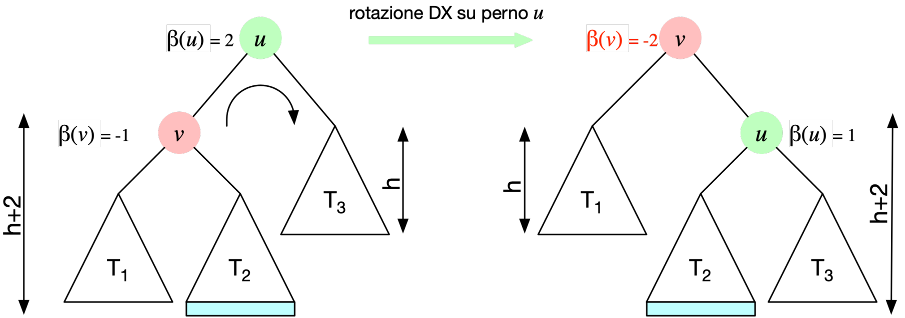




Sbilanciamento SD ⇒ Rotazione DX?

- Ribilanciamento per uno sbilanciamento SD con Rotazione DX

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



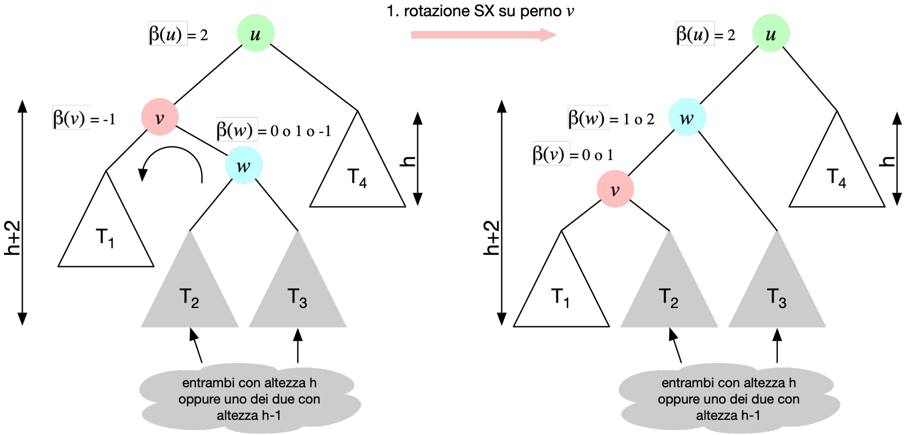
- Non funziona!!
- Abbiamo ottenuto uno sbilanciamento DS

Sbilanciamento SD ⇒ Rotazione SXDX 1/2

- Ribilanciamento per uno sbilanciamento SD ⇒ Rotazione SXDX

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

- Step 1: rotazione SX con perno *u.left*

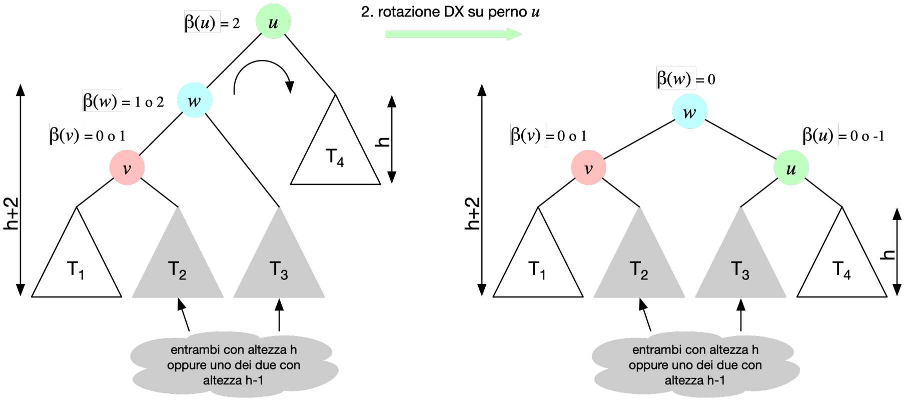


Sbilanciamento SD ⇒ Rotazione SXDX 2/2

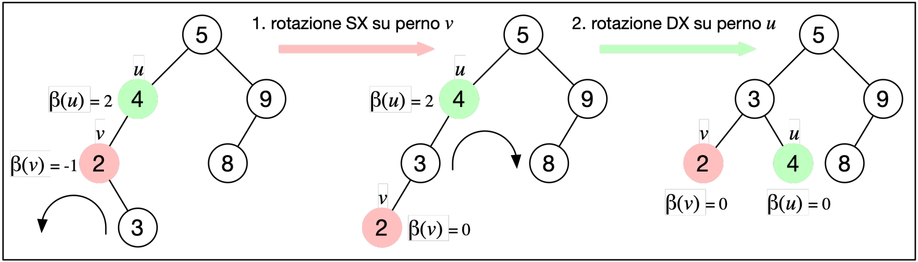
- Ribilanciamento per uno sbilanciamento SD ⇒ Rotazione SXDX

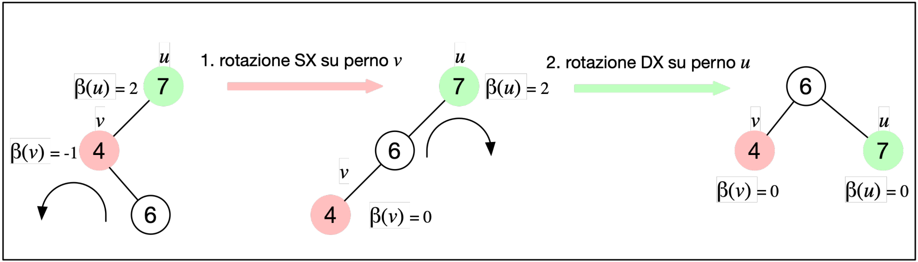
$$\beta(u) = 2 \text{ e } \beta(u.left) = -1$$

- Step 2: rotazione DX con perno u



Esempi: sbilanciamento SD ⇒ 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

PSEUDOCODICE: BILANCIAMENTO

```
function BALANCE(AVL T, NODE u)
    if u ≠ NIL and |β(u)| == 2 then
        if β(u) == 2 then
            if β(u.left) == −1 then        ▷ Sbilanciamento SD
                ROTATESX(T, u.left)
            ROTATEDX(T, u)
        else
            if β(u.right) == 1 then        ▷ Sbilanciamento DS
                ROTATEDX(T, u.right)
            ROTATESX(T, u)
```

- Costo: $O(1)$

Java (asdlab.libreria.AlberiRicerca.AlberoAVL)

Metodo per il ribilanciamento di un albero AVL tramite rotazione

```java
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: $O(\log n)$

Java (asdlab.libreria.AlberiRicerca.AlberoAVL)

Metodo per l'inserimento in un albero AVL

```java
1  public Rif insert(Object e, Comparable k) {
2      InfoAVL i = new InfoAVL(e, k);
3      Nodo v = super.insert(i);
4      Nodo p = alb.padre(v);
5      while (p != null) {
6        if (bil(p)==-2 || bil(p)==2) break;
7        aggiornaAltezza(p);
8        p = alb.padre(p);
9      }
10     if (p != null) ruota(p);
11     return i;
12 }
```

- Aggiorniamo l'altezza nel percorso nodo inserito-radice e ci fermiamo se troviamo un nodo sbilanciato (ciclo while a riga 5)
- Se troviamo un nodo sbilanciato chiamiamo la procedura di rotazione su tale nodo (riga 10)

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: $O(\log n)$

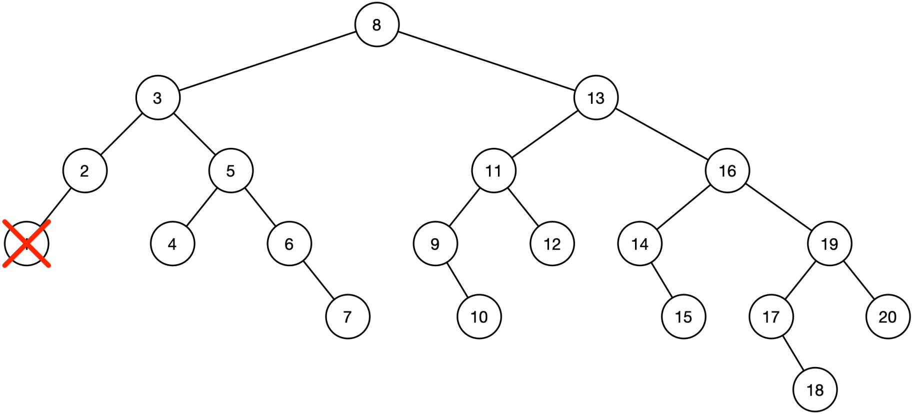
3. Se un nodo presenta fattore di bilanciamento ± 2 (nodo critico), occorre ribilanciare l'albero con la procedura di ribilanciamento
   - N.B. In caso di rimozione potremmo avere più nodi critici
   - Tutti i nodi critici sono in un unico percorso radice-foglia
   - Costo nel caso pessimo: $O(\log n)$

- Costo della rimozione in un albero AVL nel caso pessimo: $O(\log n)$

Metodo per la rimozione in un albero AVL

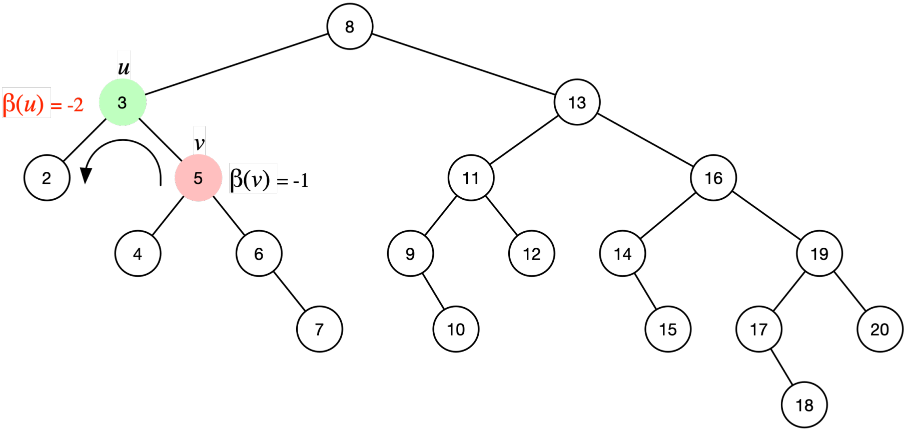
```
public void delete(Rif i) {
    Nodo p = super.delete((InfoAVL)i);
    while (p != null) {
      if (bil(p)==-2 || bil(p)==2) ruota(p);
      else aggiornaAltezza(p);
      p = alb.padre(p);
    }
}
```

Esempio: rotazioni a cascata 1/4



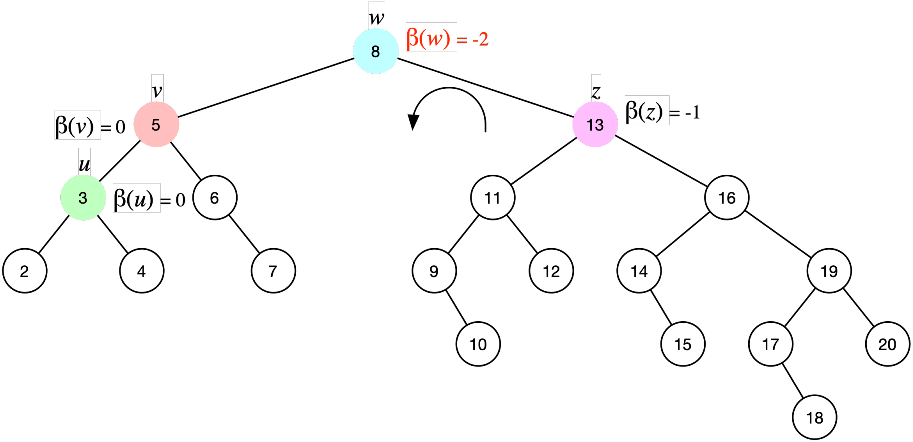
Albero AVL bilanciato prima della rimozione del nodo con chiave 1

ESEMPIO: ROTAZIONI A CASCATA 2/4



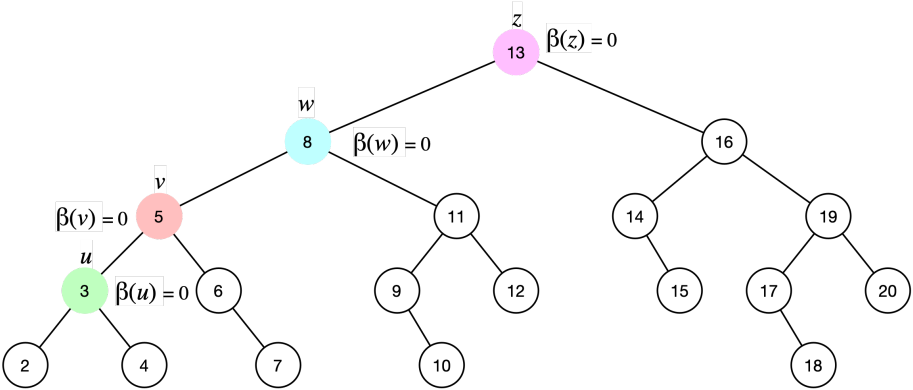
Ribilanciamento: rotazione SX con perno u

Esempio: rotazioni a cascata 3/4



Ribilanciamento: rotazione SX con perno *w*

Esempio: rotazioni a cascata 4/4



Albero ribilanciato

Altri alberi bilanciati

- Gli Alberi AVL non sono l'unica struttura dati auto-bilanciante
- Altre implementazioni non richiedono necessariamente alberi binari
- Tra le strutture dati maggiormente note abbiamo
  - B-alberi
    - Alberi generici, non necessariamente binari
    - Utilizzati per database e file system
  - Alberi 2-3
    - Ogni nodo intero ha 2 oppure 3 nodi
    - Tutte le foglie sono sempre allo stesso livello
    - Le chiavi sono tutte nelle foglie (e ordinate)
  - RB-Alberi (Red-Back Trees)
    - Alberi binari come gli AVL
- Tutte queste strutture supportano le operazioni di inserimento, rimozione e ricerca in tempo logaritmico

Dizionario: riassunto dei costi

	SEARCH		INSERT		DELETE	
	Medio	Pessimo	Medio	Pessimo	Medio	Pessimo
Array non ordinati	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$\Theta(n)$	$\Theta(n)$
Array ordinati	$O(\log n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Lista concatenata	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Albero Binario di Ricerca	$O(\overline{h})$	$O(h)$	$O(\overline{h})$	$O(h)$	$O(\overline{h})$	$O(h)$
Albero AVL	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

- $h =$ altezza dell'albero, $\overline{h} =$ altezza media dell'albero
- Gli Alberi AVL ci assicurano un costo pessimo e medio logaritmico per tutte le operazioni