Algoritmi e Strutture Dati

Capitolo 6

Il problema del Dizionario

il problema del dizionario

tipo `Dizionario`:
dati: un insieme S di coppie $(elem, chiave)$
operazioni:

`insert`$(elem\ e, chiave\ k)$
aggiunge a S una nuova coppia (e, k)

`delete`$(elem\ e)$
cancella da S l'elemento e

`search`$(chiave\ k) \rightarrow elem$
se la chiave k è presente in S restituisce un elemento e ad essa associato, e `null` altrimenti

Come implementare efficientemente un dizionario?

è possibile garantire che tutte le operazioni su un dizionario di n elementi abbiano tempo O(log n).

Due idee:

Definire un albero (binario) tale che ogni operazione richiede tempo O(altezza albero)

alberi binari di ricerca

fare in modo che l'altezza dell'albero sia sempre O(log n)

alberi AVL

Alberi binari di ricerca
(BST = binary search tree)

Definizione

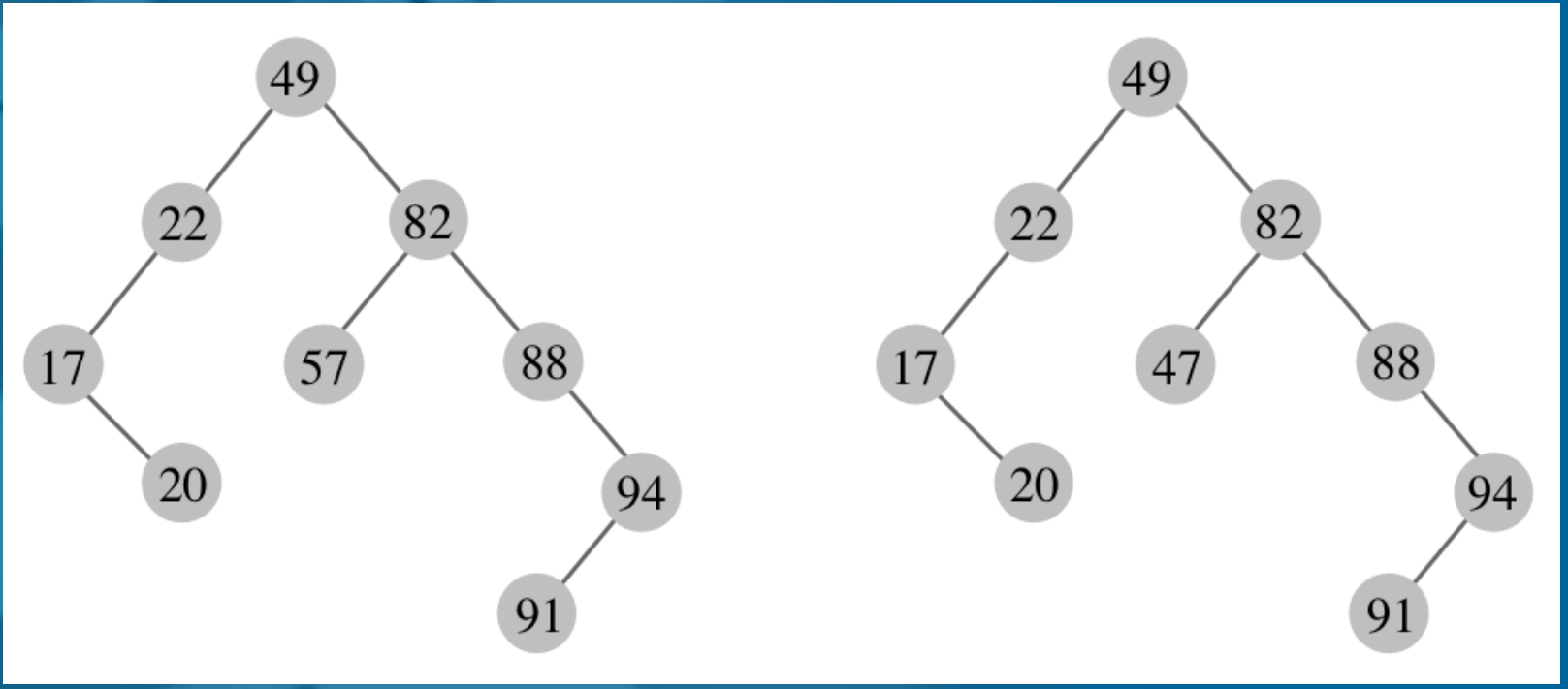
Albero binario che soddisfa le seguenti proprietà

- ogni nodo *v* contiene un elemento elem(v) cui è associata una chiave chiave(v) presa da un dominio totalmente ordinato.

Per ogni nodo v vale che:

- le chiavi nel sottoalbero sinistro di v sono ≤ chiave(*v*)
- le chiavi nel sottoalbero destro di v sono > chiave(*v*)

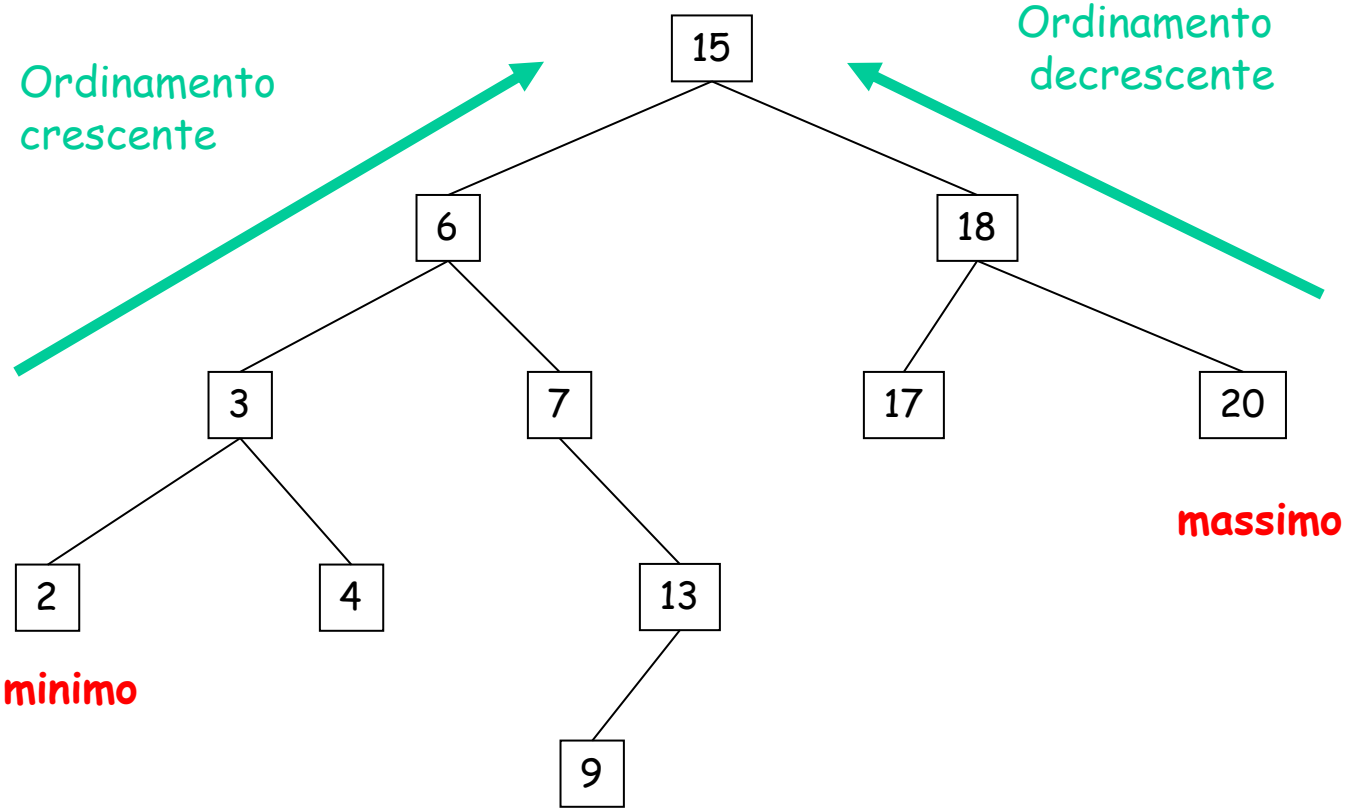
Esempi



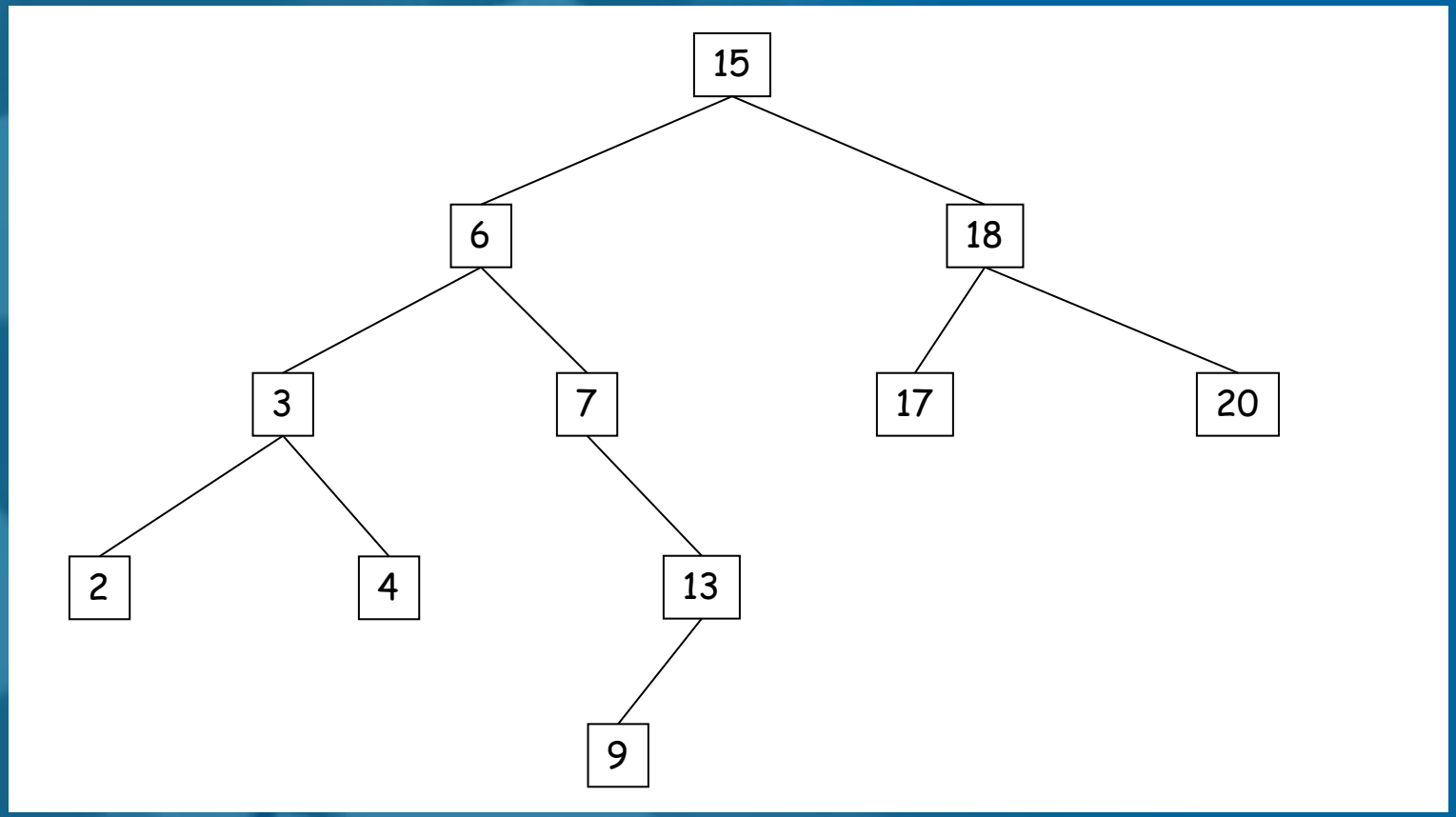
Albero binario
di ricerca

Albero binario
non di ricerca

…ancora un esempio…



…che succede se visitiamo un BST in ordine simmetrico?



2 3 4 6 7 9 13 15 17 18 20

visito i nodi in ordine crescente di chiave!

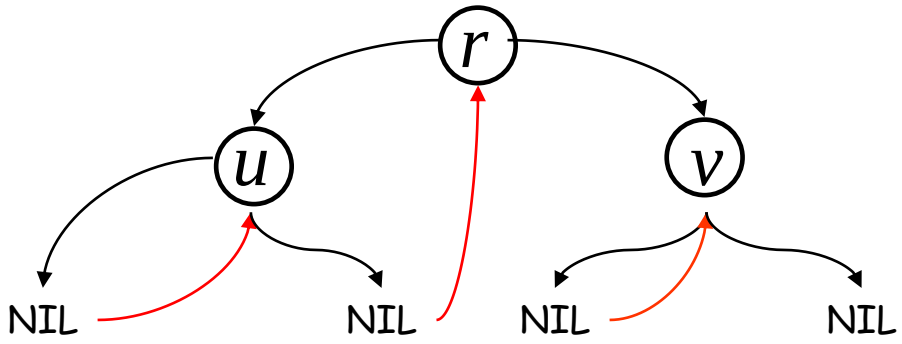
Verifica di correttezza –

Indichiamo con h l'altezza dell'albero.

Vogliamo mostrare che la visita in ordine simmetrico restituisce la sequenza ordinata

Per induzione sull'altezza dell'ABR: h=1

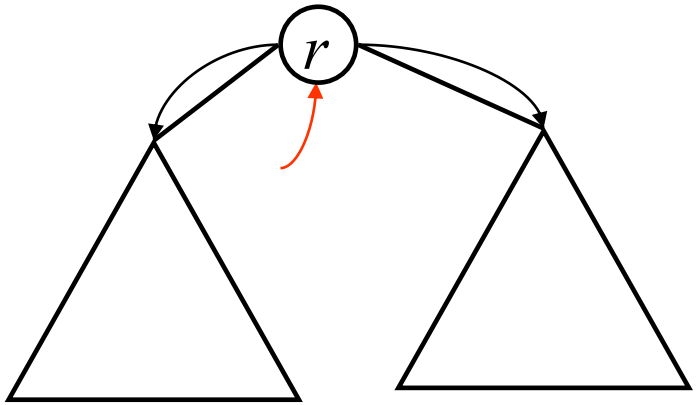
(mostriamolo senza perdita di generalità quando l'albero è completo.)



$$\text{chiave}(u) \leq \text{chiave}(r) \leq \text{chiave}(v)$$

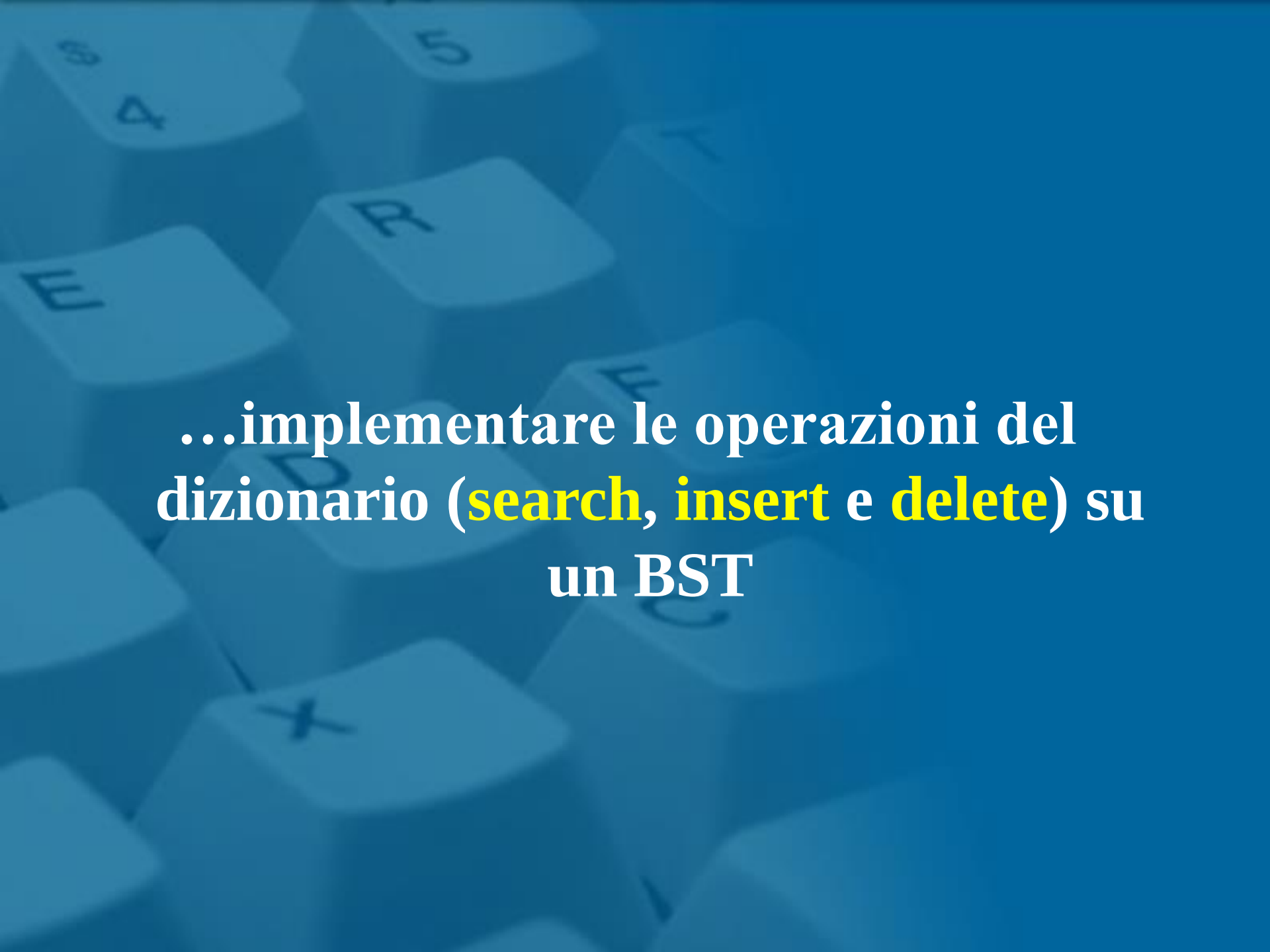
Verifica correttezza (continua …)

h = generico (ipotizzo che la procedura sia corretta per altezza <h)



Albero di altezza ≤ h-1. Tutti i suoi elementi sono minori o uguali della radice

Albero di altezza ≤ h-1. Tutti i suoi elementi sono maggiori o uguali della radice



…implementare le operazioni del dizionario (search, insert e delete) su un BST

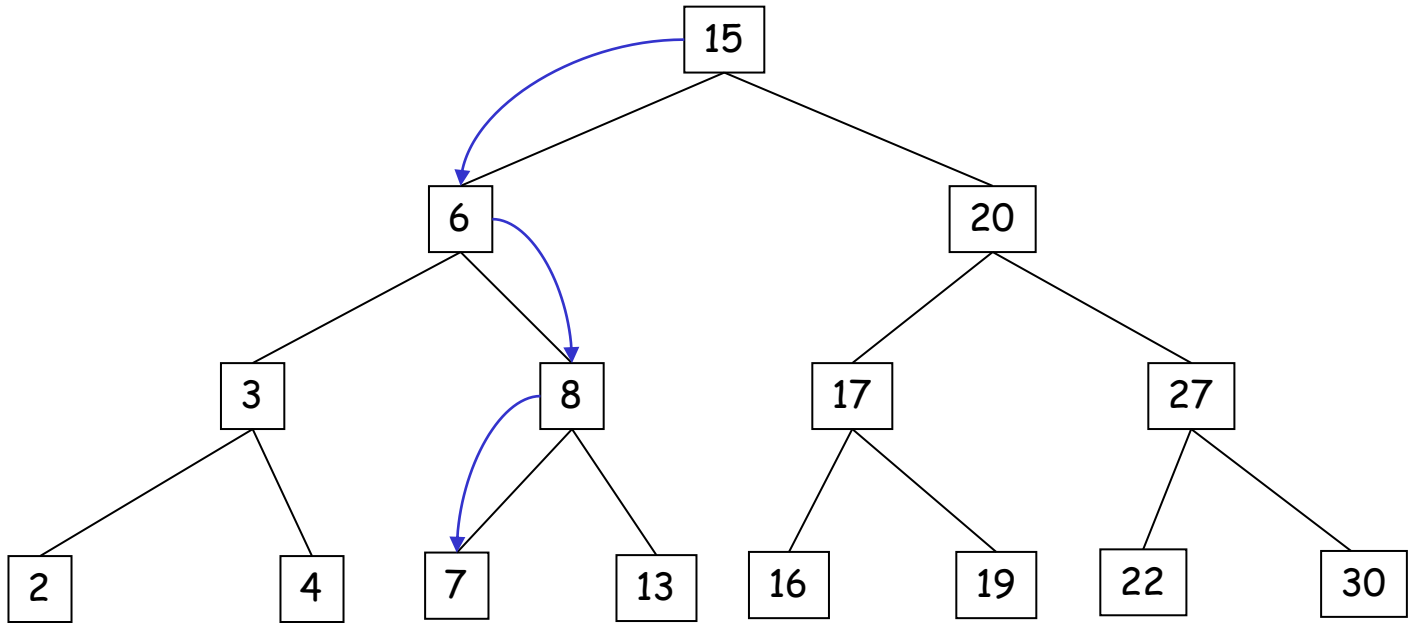
search(chiave k) -> elem

Traccia un cammino nell'albero partendo dalla radice: su ogni nodo, usa la proprietà di ricerca per decidere se proseguire nel sottoalbero sinistro o destro

algoritmo `search`$(chiave\ k) \rightarrow elem$

1. $v \leftarrow$ radice di T
2. **while** ($v \neq$ `null`) **do**
3. **if** ($k = chiave(v)$) **then return** $elem(v)$
4. **else if** ($k < chiave(v)$) **then** $v \leftarrow$ figlio sinistro di v
5. **else** $v \leftarrow$ figlio destro di v
6. **return** `null`

search(7)

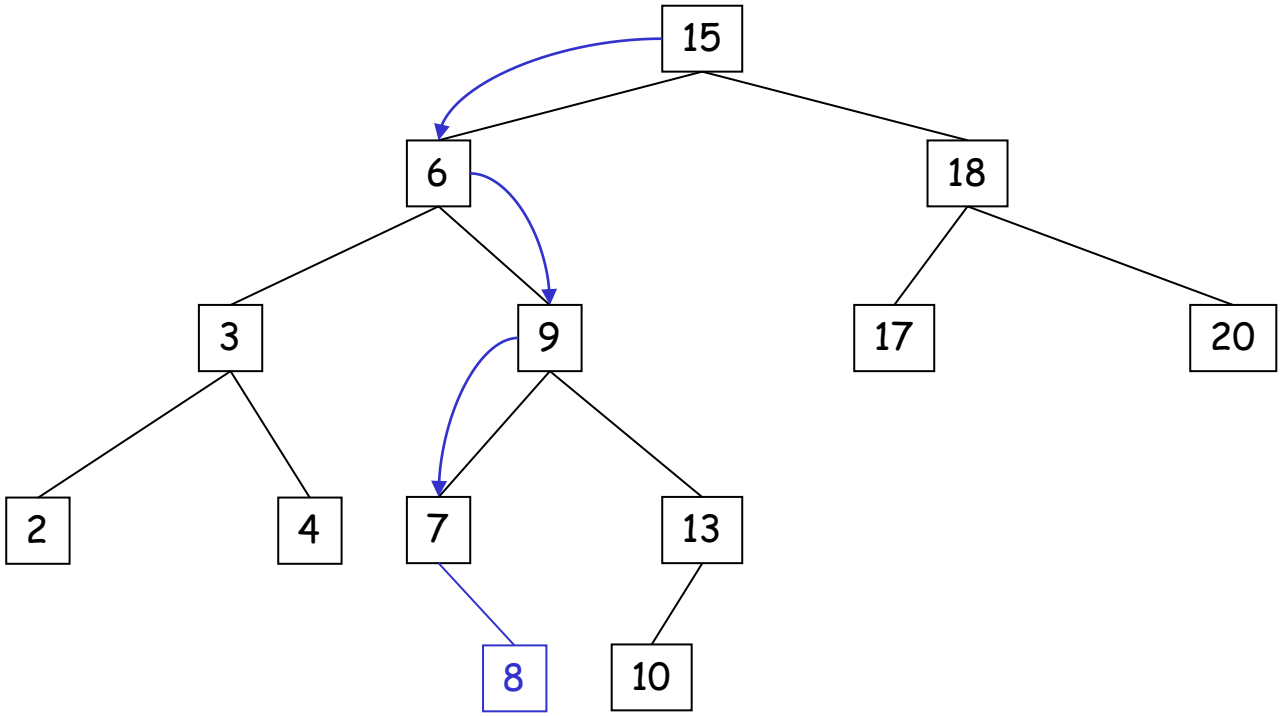


insert(elem e, chiave k)

Idea: aggiunge la nuova chiave come nodo foglia;
per capire dove mettere la foglia simula una ricerca con la chiave da inserire

1. Crea un nuovo nodo u con elem=e e chiave=k
2. Cerca la chiave k nell'albero, identificando così il nodo v che diventerà padre di u
3. Appendi u come figlio sinistro/destro di v in modo che sia mantenuta la proprietà di ricerca

insert(e,8)



correttezza:

Se seguo questo schema l'elemento *e* viene posizionato nella posizione giusta. Infatti, per costruzione, ogni antenato di *e* si ritrova *e* nel giusto sottoalbero.

…qualche operazione ausiliaria prima di implementare l'operazione di delete:

…massimo, minimo,

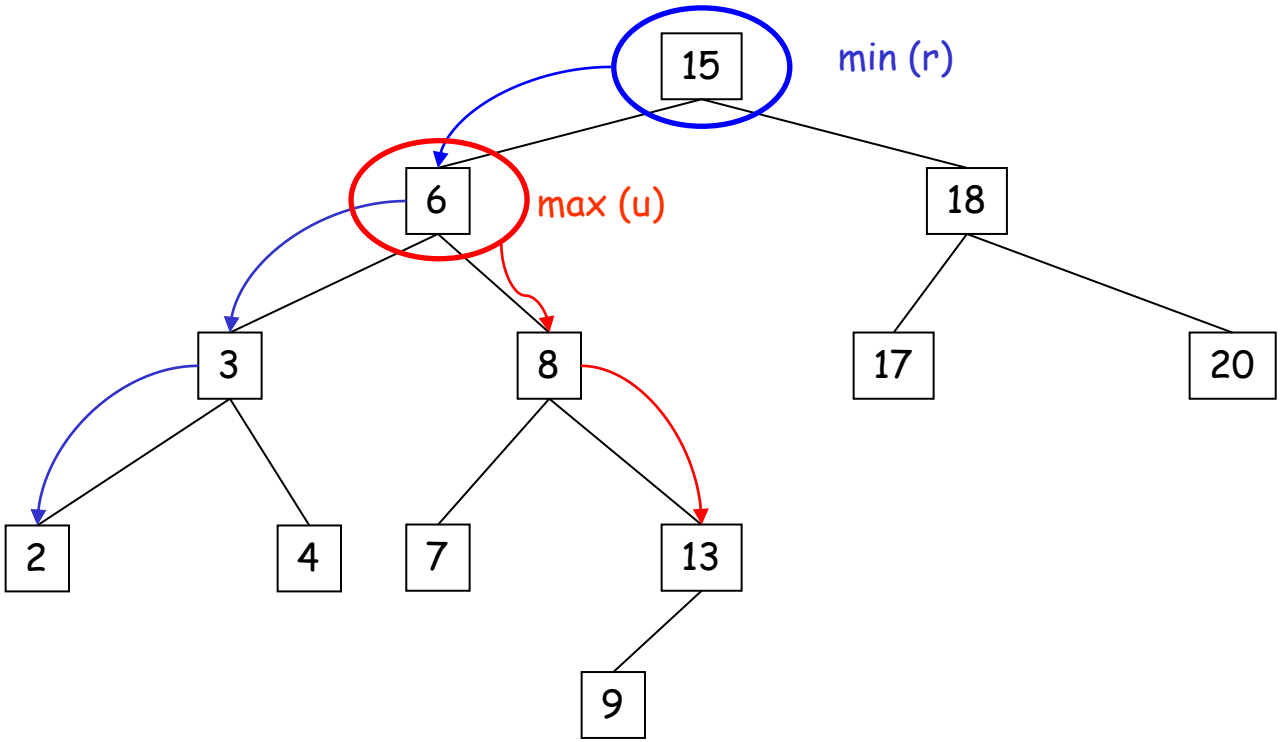
predecessore e successore

Ricerca del massimo

algoritmo `max`$(nodo\ u) \rightarrow nodo$

1. $v \leftarrow u$
2. **while** (figlio destro di $v \neq$ `null`) **do**
3. $v \leftarrow$ figlio destro di v
4. **return** v

Nota: è possibile definire una procedura `min`(nodo *u*) in maniera del tutto analoga


15
min (r)
6
max (u)
18
3
8
17
20
2
4
7
13
9

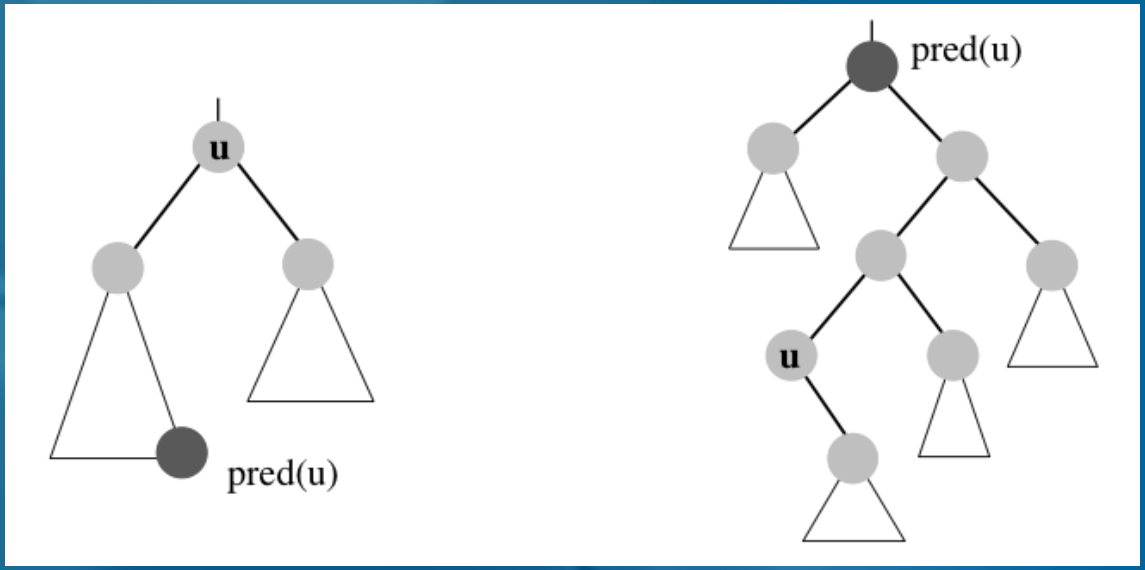
predecessore e successore

- il predecessore di un nodo *u* in un BST è il nodo *v* nell'albero avente massima chiave ≤ chiave(*u*)
- il successore di un nodo *u* in un BST è il nodo *v* nell'albero avente minima chiave ≥ chiave(*u*)
- Come trovo il predecessore/successore di un nodo in un BST?

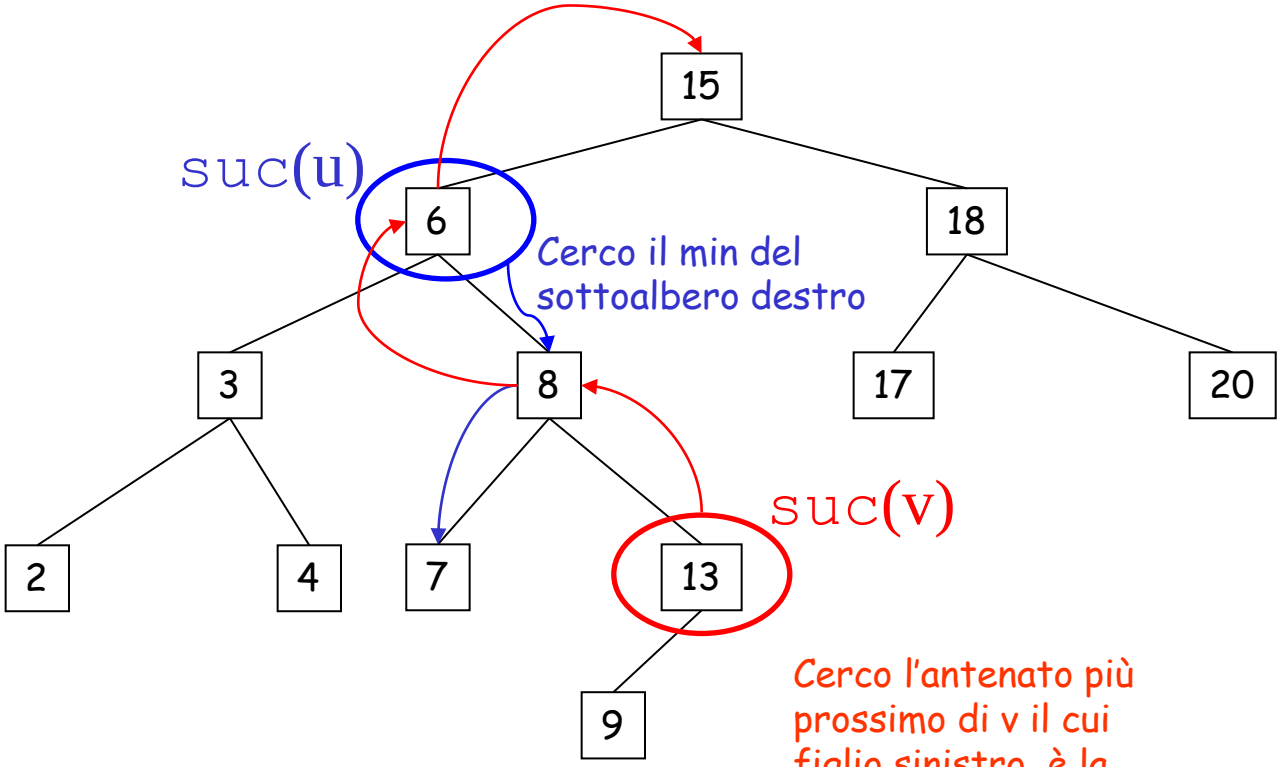
Ricerca del predecessore

algoritmo `pred`$(nodo\ u) \rightarrow nodo$

1. **if** (u ha figlio sinistro $sin(u)$) **then**
2. **return** `max`$(sin(u))$
3. **while** ($parent(u) \neq null$ e u è figlio sinistro di suo padre) **do**
4. $u \leftarrow parent(u)$
5. **return** $parent(u)$



Nota: la ricerca del **successore** di un nodo è simmetrica

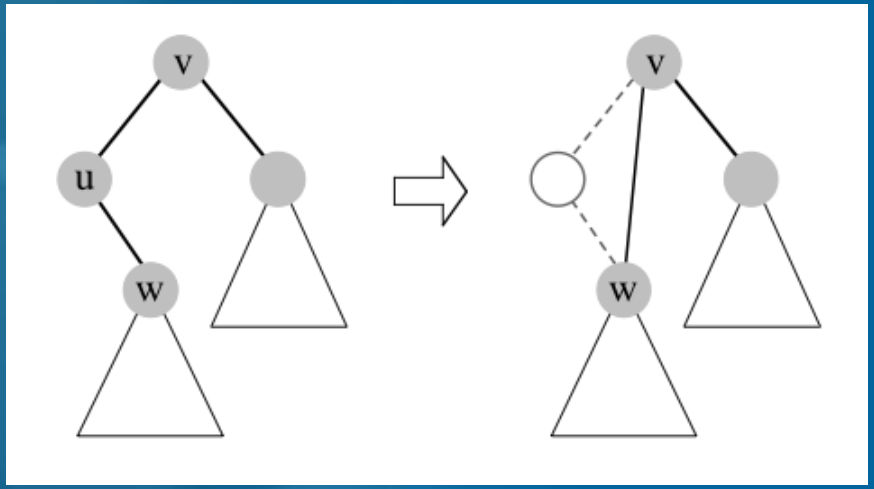


delete(elem e)

Sia u il nodo contenente l'elemento e da cancellare:

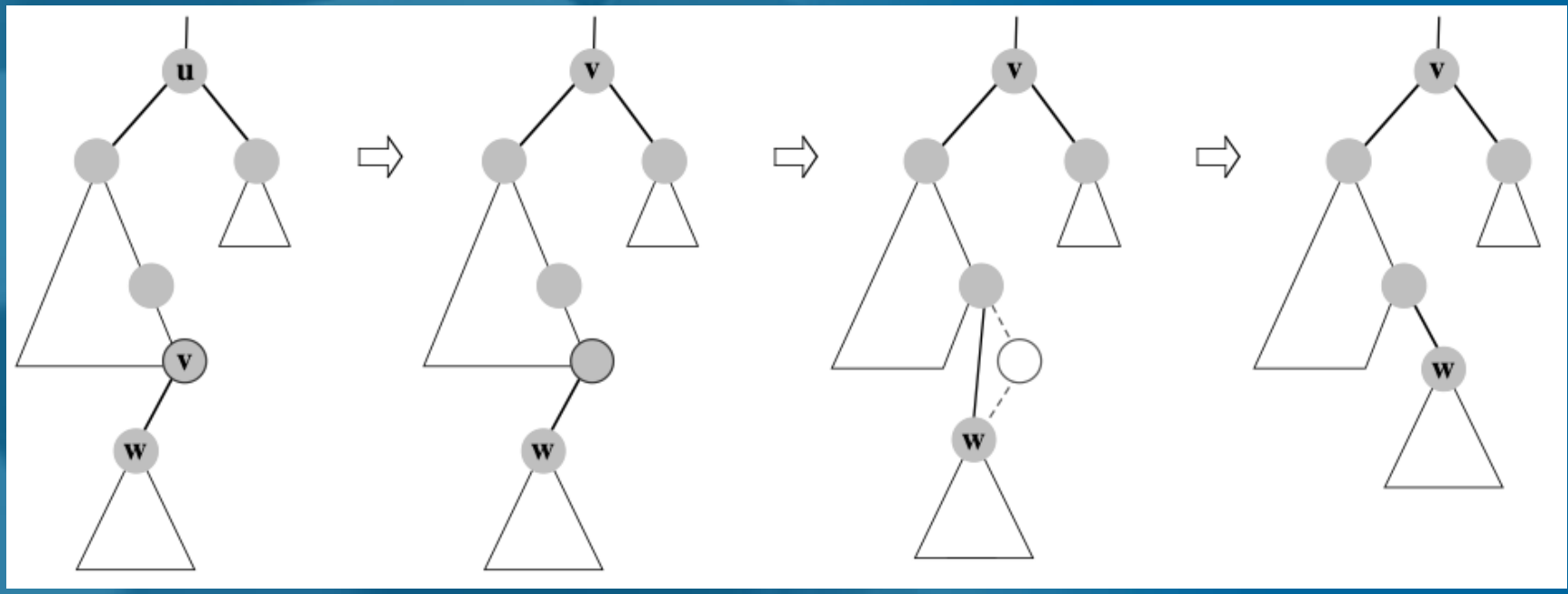
1) u è una foglia: rimuovila

2) u ha un solo figlio:

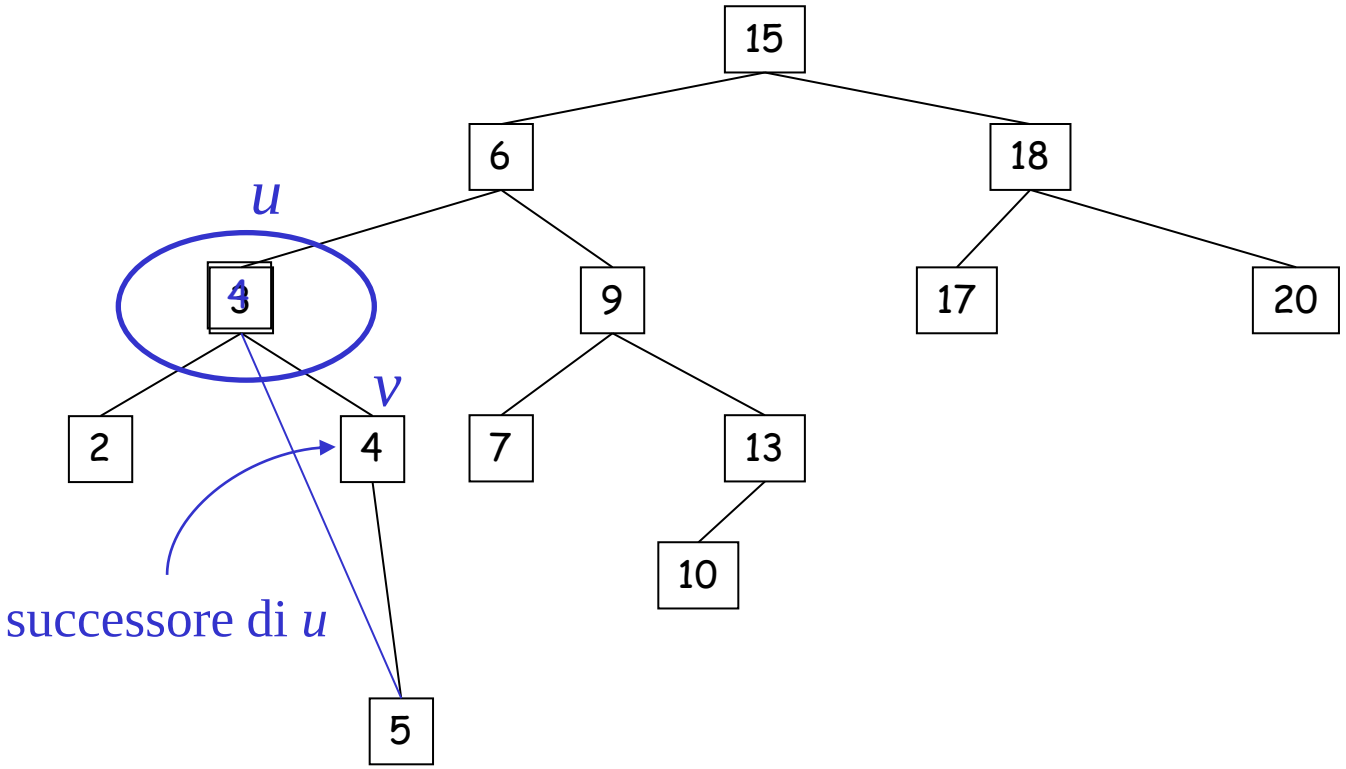


delete(elem e)

3) u ha due figli: sostituiscilo con il predecessore (o successore) (v) e rimuovi fisicamente il predecessore (o successore) (che ha al più un figlio)



delete (u)

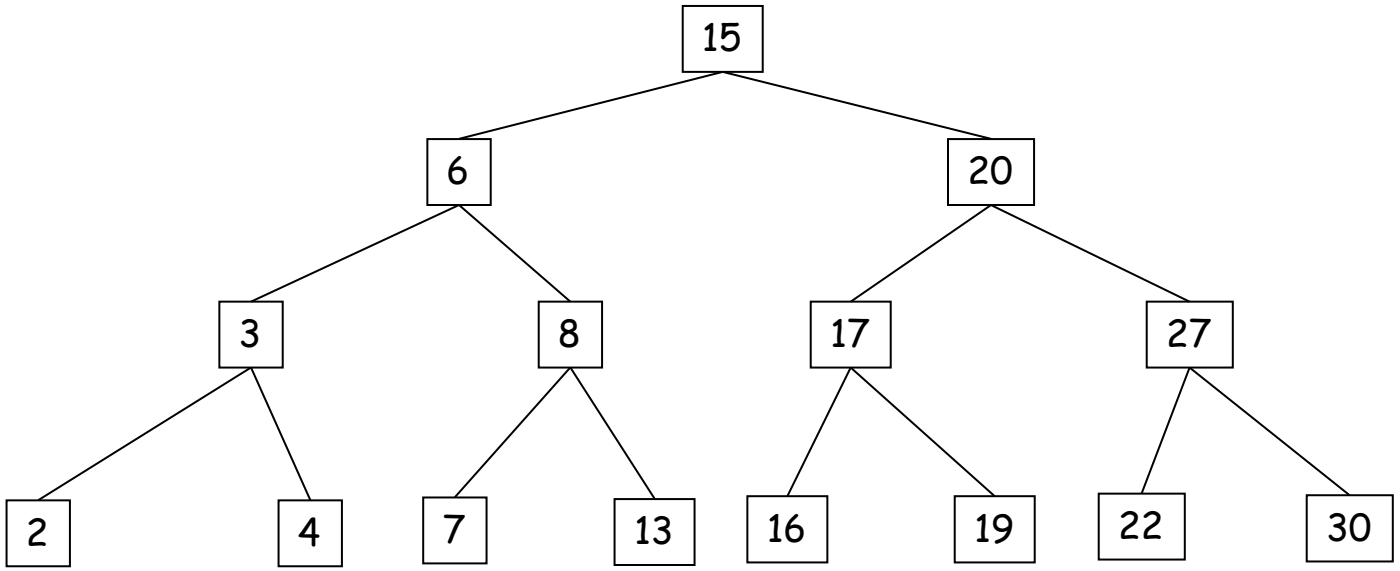


Costo delle operazioni

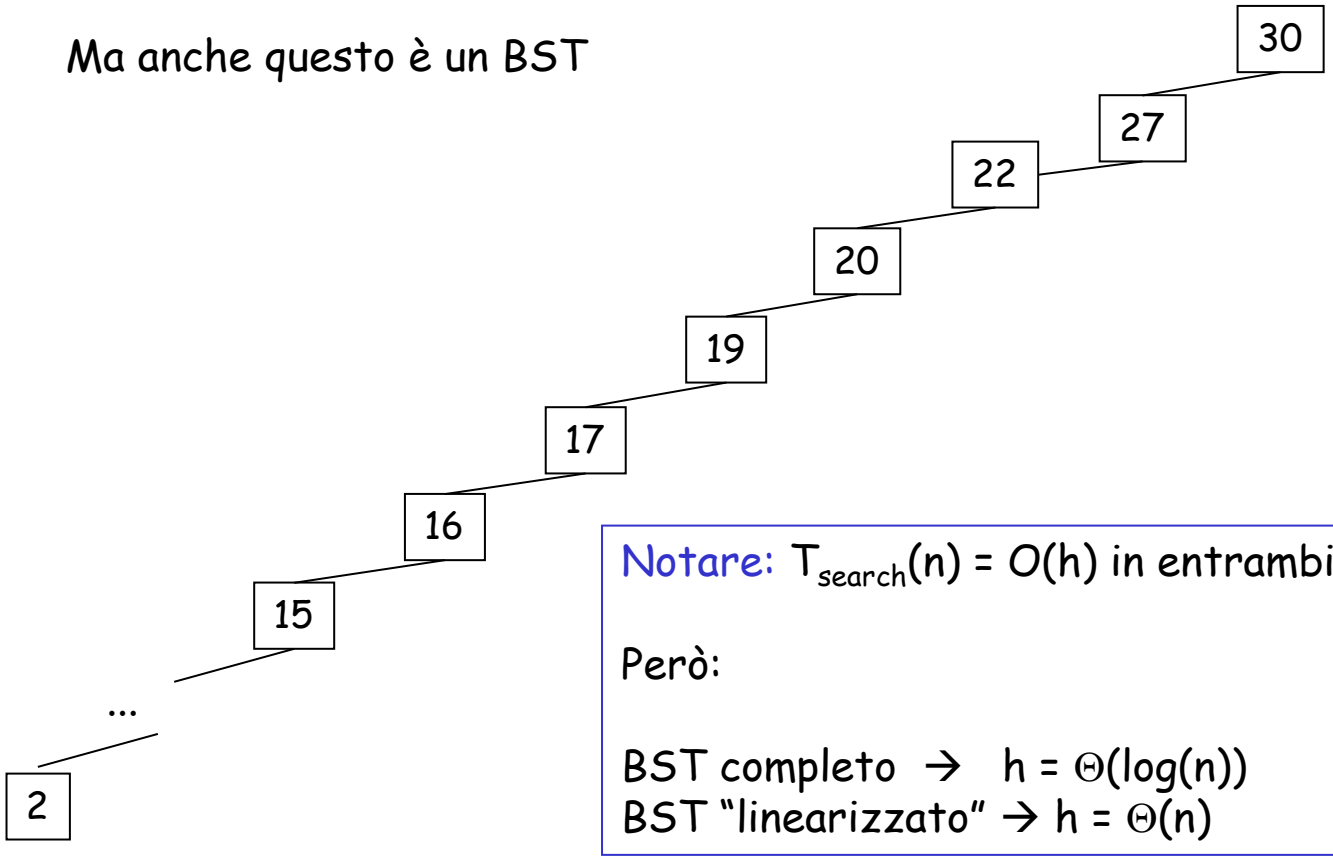
- Tutte le operazioni hanno costo O(h) dove h è l'altezza dell'albero
- O(n) nel caso peggiore (alberi molto sbilanciati e profondi)

…un albero binario di ricerca bilanciato…

$h=O(\log n)$



Ma anche questo è un BST



Notare: $T_{search}(n) = O(h)$ in entrambi i casi

Però:

BST completo → $h = \Theta(\log(n))$
BST "linearizzato" → $h = \Theta(n)$

il problema del dizionario

tipo `Dizionario`:
dati: un insieme S di coppie $(elem, chiave)$
operazioni:

`insert`$(elem\ e, chiave\ k)$
aggiunge a S una nuova coppia (e, k)

`delete`$(elem\ e)$
cancella da S l'elemento e

`search`$(chiave\ k) \rightarrow elem$
se la chiave k è presente in S restituisce un elemento e ad essa associato, e `null` altrimenti

Come implementare efficientemente un dizionario?

è possibile garantire che tutte le operazioni su un dizionario di n elementi abbiano tempo O(log n).

Due idee:

Definire un albero (binario) tale che ogni operazione richiede tempo O(altezza albero)

alberi binari di riceca

fare in modo che l'altezza dell'albero sia sempre O(log n)

alberi AVL

Alberi AVL
(Adel’son-Vel’skii e Landis, 1962)

Definizioni

Fattore di bilanciamento $\beta(v)$ di un nodo v = altezza del sottoalbero sinistro di v - altezza del sottoalbero destro di v

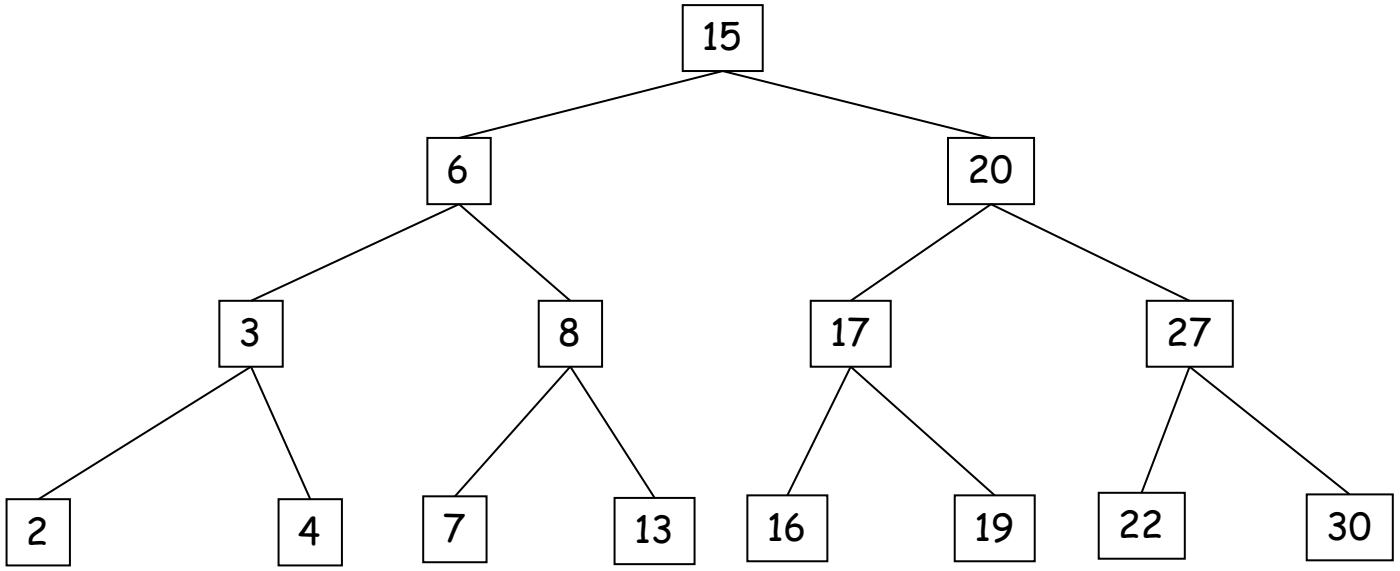
Un albero si dice bilanciato in altezza se ogni nodo v ha fattore di bilanciamento in valore assoluto ≤ 1

Alberi AVL = alberi binari di ricerca bilanciati in altezza

Generlemente $\beta(v)$ mantenuto come informazione addizionale nel record relativo a v

…qualche esempio…

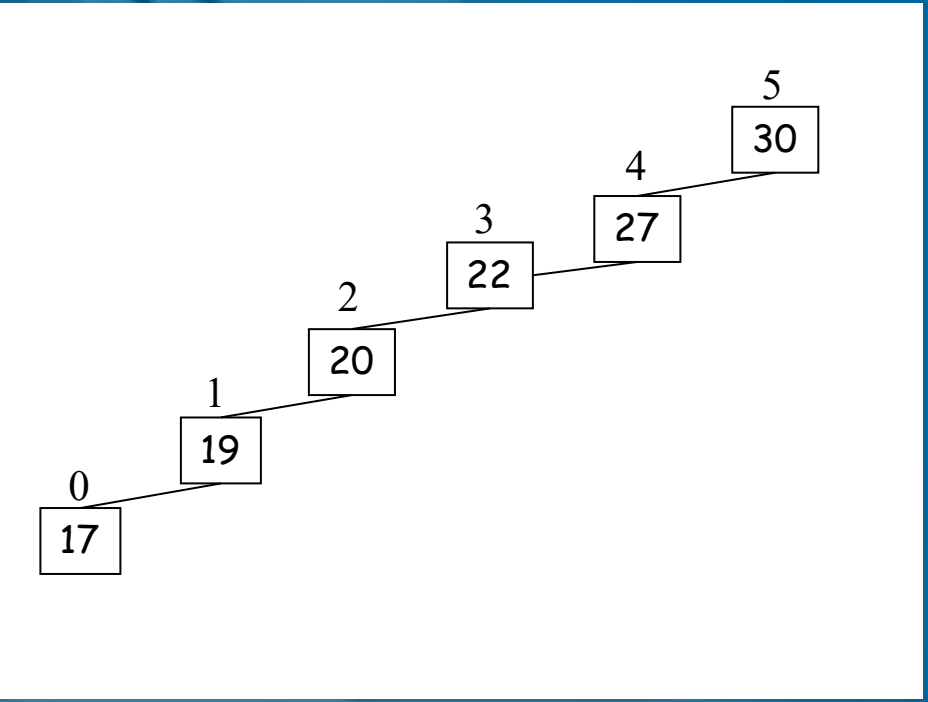
è il seguente albero AVL?



Sì: tutti i nodi hanno fattore di bilanciamento = 0

…qualche esempio…

è il seguente albero AVL?

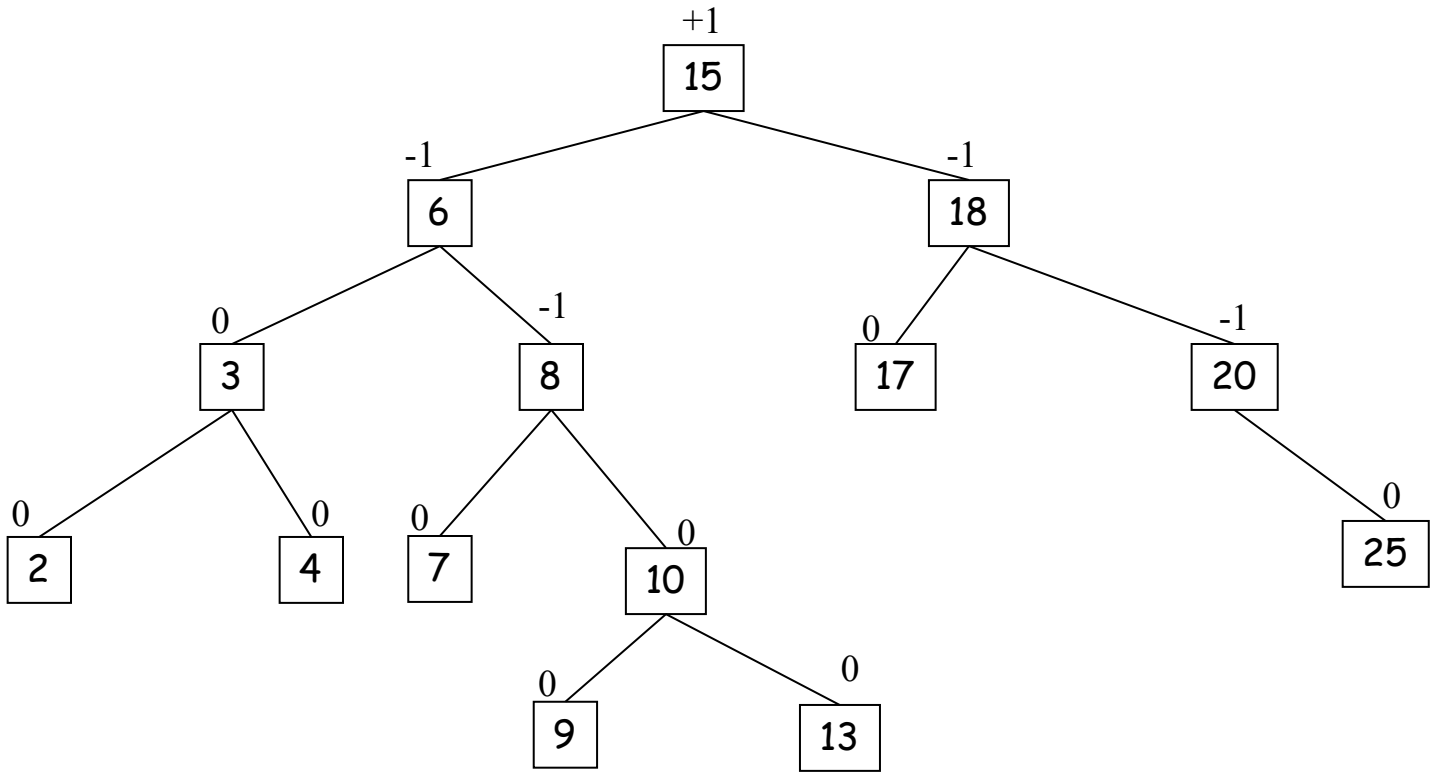


Convenzione:
altezza di un albero vuoto= -1

NO! Non vale la proprietà sui fattori di bilanciamento!

è il seguente albero AVL?

…qualche esempio…



Sì: proprietà sui fattori di bilanciamento rispettata

Altezza di alberi AVL

Si può dimostrare che un albero AVL con n nodi ha altezza O(log n)

Idea della dimostrazione: considerare, tra tutti gli AVL, i più sbilanciati

albero di Fibonacci di altezza h:

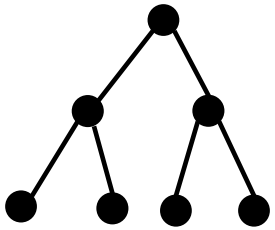
albero AVL di altezza h con il minimo numero di nodi n_h

minimizzare # nodi fissata l'altezza $\equiv$ massimizzare altezza fissato #nodi

Intuizione: se gli alberi di Fibonacci hanno altezza O(log n), allora tutti gli alberi AVL hanno altezza O(log n)

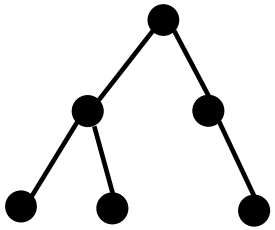
Un esempio: come è fatto un albero di Fibonacci di altezza 2?

così?



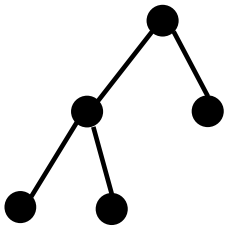
no: troppi nodi!

così?



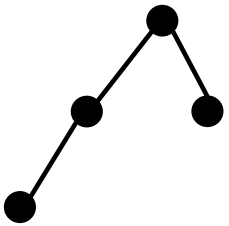
no: troppi nodi!

così?



no: troppi nodi!

così?



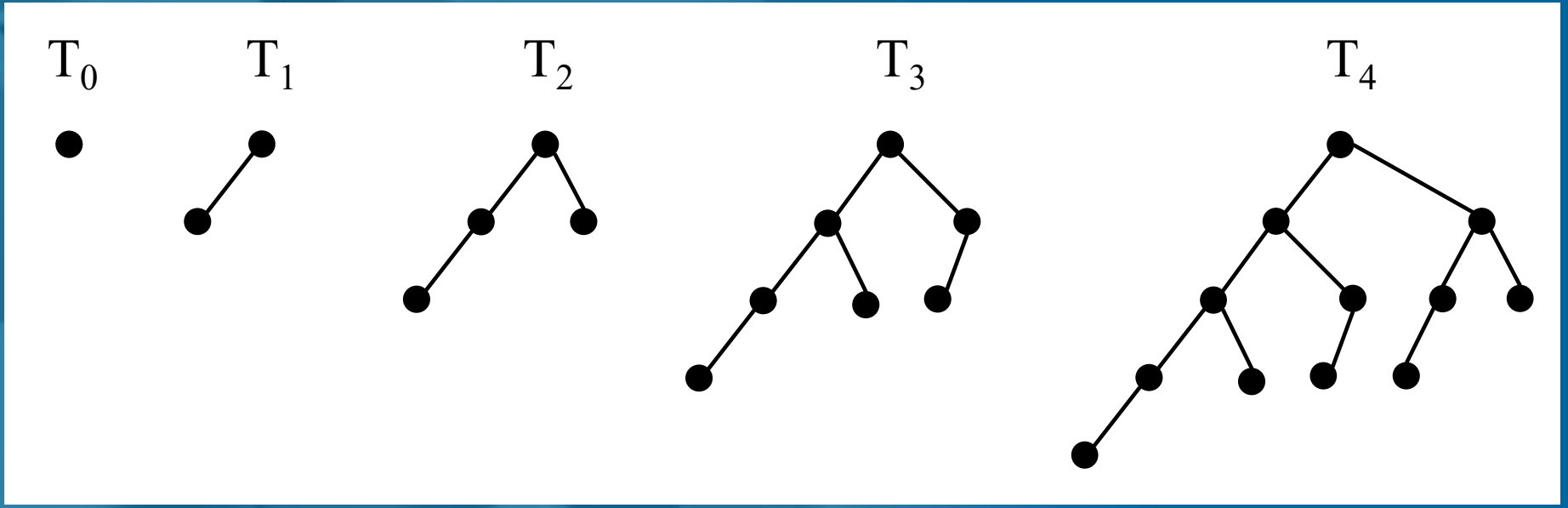
sì!

Infatti: se togliamo ancora un nodo, o diventa sbilanciato, o cambia la sua altezza

Nota: ogni nodo (non foglia) ha fattore di bilanciamento pari (in valore assoluto) a 1

…Alberi di Fibonacci per valori piccoli di altezza…

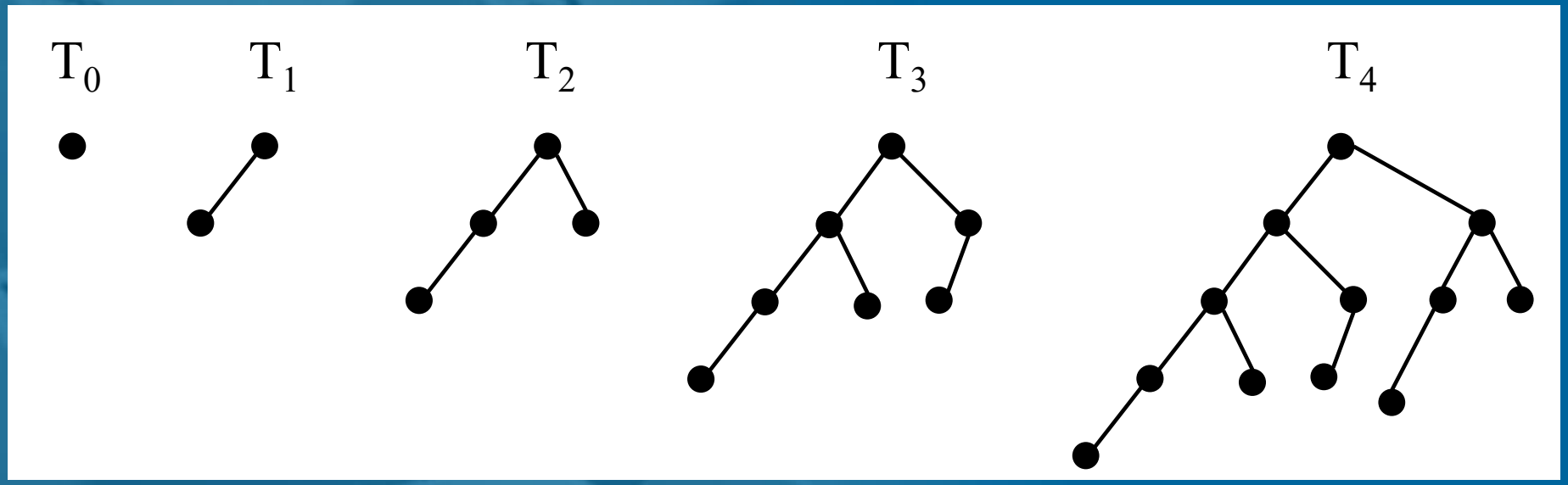
T_i: albero di Fibonacci di altezza i
(albero AVL di altezza i con il minimo numero di nodi)



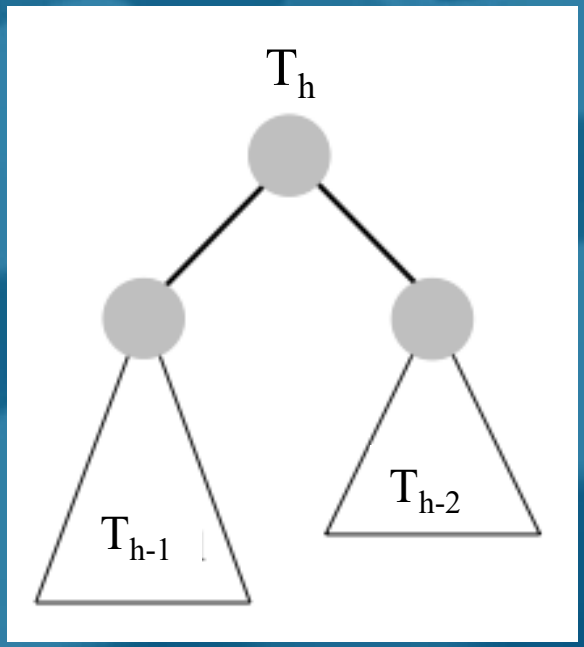
Nota che: se a T_i tolgo un nodo, o diventa sbilanciato, o cambia la sua altezza

Inoltre: ogni nodo (non foglia) ha fattore di bilanciamento pari (in valore assoluto) a 1

intravedete uno schema per generare l'i-esimo albero di Fibonacci a partire dai precedenti?



Lo schema



Lemma

Sia n_h il numero di nodi di T_h.

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

F_i: i-esimo numero di fibonacci

dim

per induzione su h

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

Corollario

Un albero AVL con n nodi ha altezza $h=O(\log n)$

dim

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

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

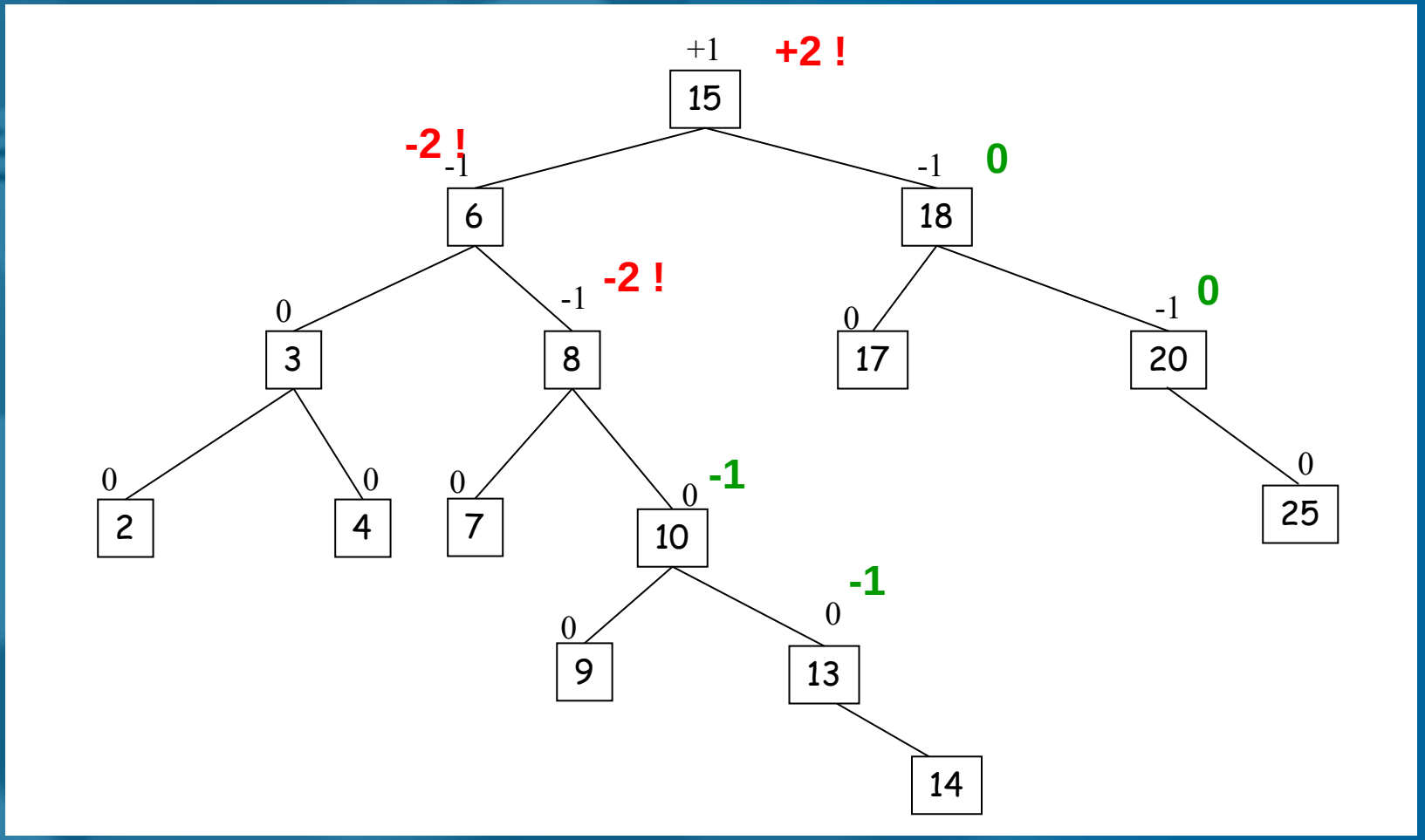
corollario segue da $n_h \leq n$

Ricorda che vale:

$F_k = \Theta(\phi^k)$

$\phi = 1.618\ldots$ sezione aurea

Posso usare un albero AVL per implementare un dizionario?



come implemento Insert(14)?

…e delete(25)?

Domanda:
di quanto e quali fattori di bilanciamento cambiano a fronte di un inserimento/cancellazione?

Se parto da un albero AVL e inserisco/cancello un nodo:

- (quali) cambiano solo i fattori di bilanciamento dei nodi lungo il cammino radice-nodo inserito/cancellato

- (quanto) i fattori di bilanciamento cambiano di +/- 1

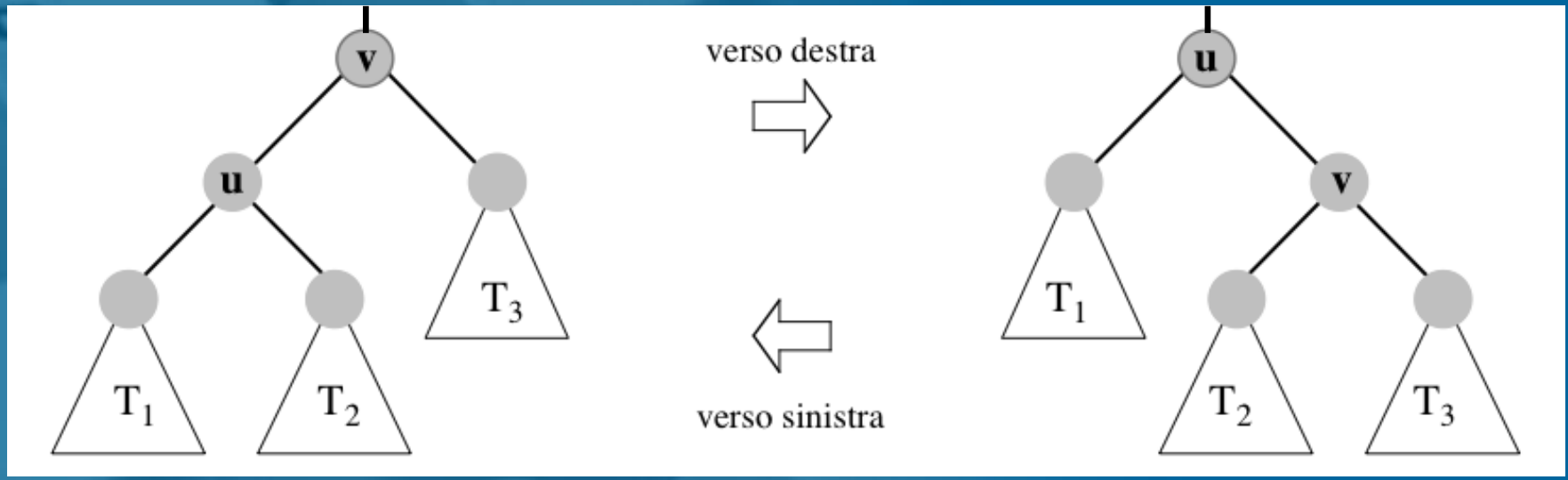
Implementazione delle operazioni

- L'operazione search procede come in un BST

- Ma inserimenti e cancellazioni potrebbero sbilanciare l'albero

$\Rightarrow$ Manteniamo il bilanciamento tramite opportune rotazioni

Rotazione di base verso destra/sinistra sul nodo v/u



- Mantiene la proprietà di ricerca
- Richiede tempo O(1)

Ribilanciamento tramite rotazioni

- Le rotazioni sono effettuate su nodi sbilanciati
- Sia v un nodo di profondità massima (nodo critico) con fattore di bilanciamento $\beta(v) \pm 2$
- Esiste un sottoalbero T di v che lo sbilancia
- A seconda della posizione di T si hanno 4 casi:

$\beta(v)=+2$ $\beta(v)=-2$

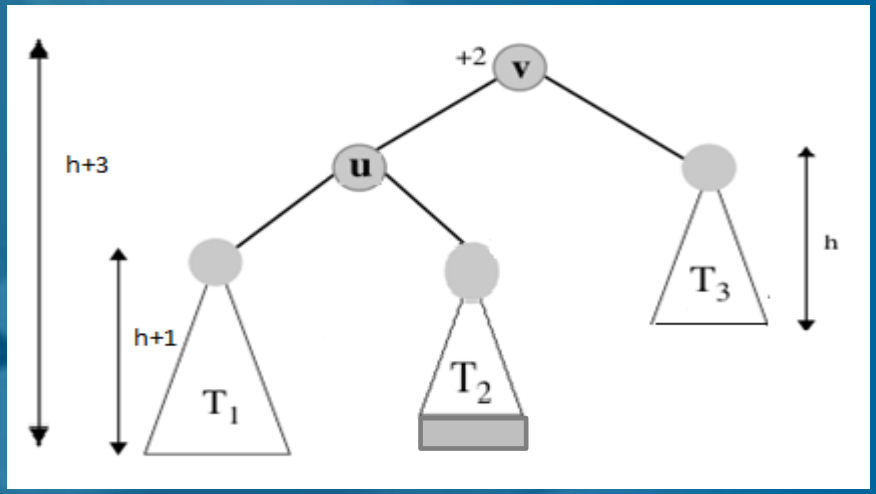
Sinistra - sinistra	(SS)	T è il sottoalbero sinistro del figlio sinistro di v
Destra - destra	(DD)	T è il sottoalbero destro del figlio destro di v
Sinistra - destra	(SD)	T è il sottoalbero destro del figlio sinistro di v
Destra - sinistra	(DS)	T è il sottoalbero sinistro del figlio destro di v

- I quattro casi sono simmetrici a coppie

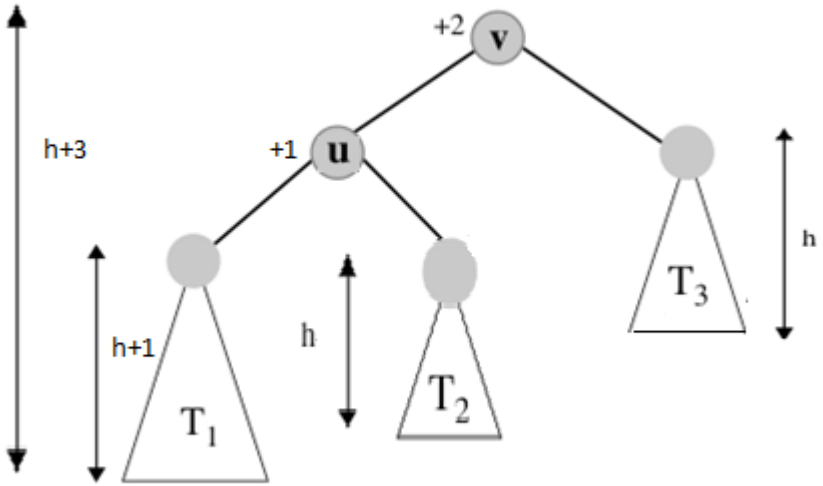
Caso SS

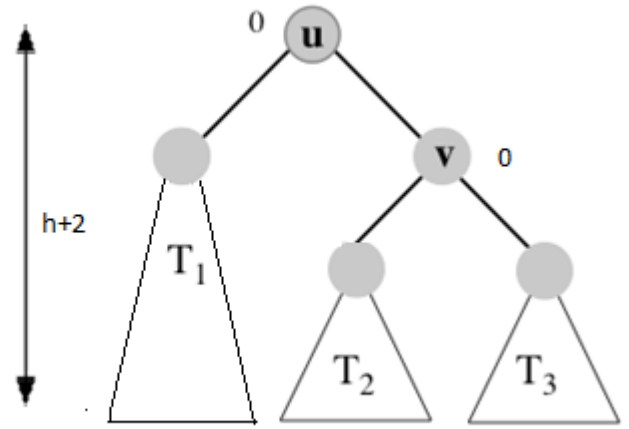
[β(v)=+2, altezza T_1=h+1]

- L'altezza di T(v) è *h*+3, l'altezza di T(u) è *h*+2, l'altezza di T_3 è *h*, e l'altezza di T_1 è *h*+1 ⇒ β(v)=+2 e lo sbilanciamento è provocato da T_1

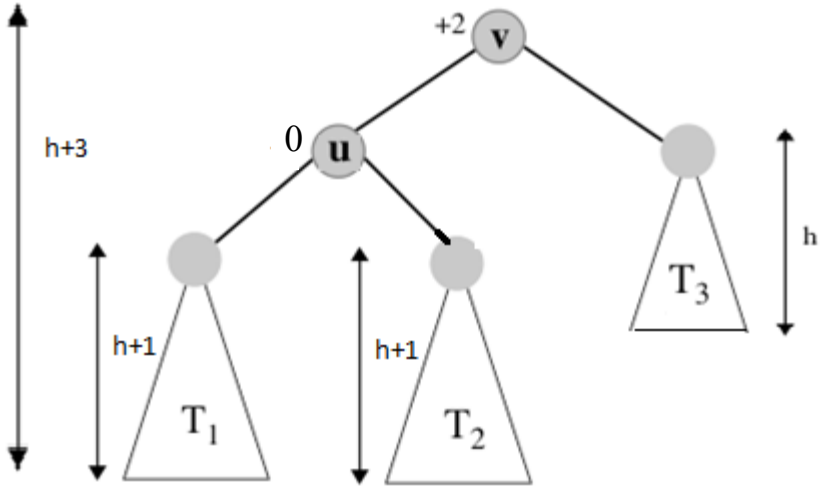


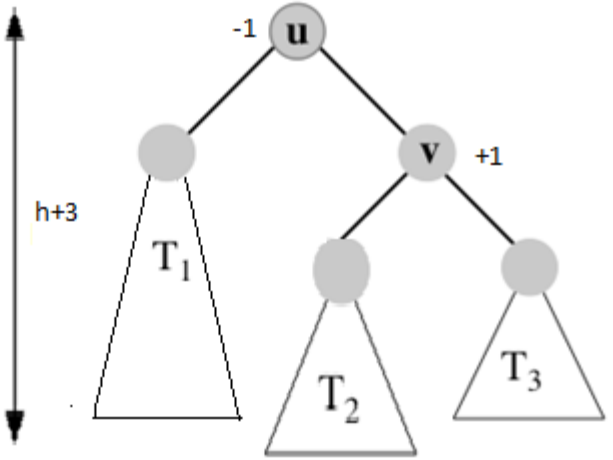
- Si applica una rotazione semplice verso destra su v; 2 sottocasi possibili:
  - (i) l'altezza di T_2 è *h* ⇒ l'altezza dell'albero coinvolto nella rotazione passa da *h*+3 a *h*+2
  - (ii) l'altezza di T_2 è *h*+1 ⇒ l'altezza dell'albero coinvolto nella rotazione rimane pari a *h*+3





…i due sottocasi del caso SS…





Osservazioni sul caso SS

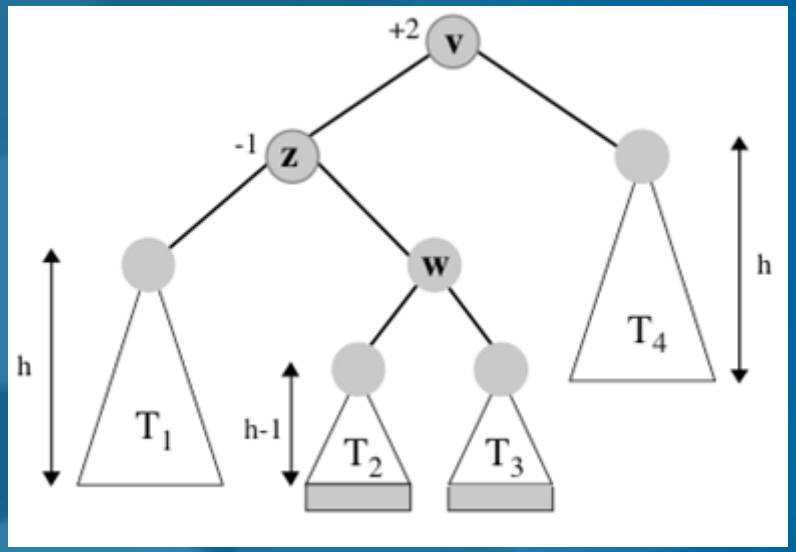
- Dopo la rotazione l'albero è bilanciato (tutti i fattori di bilanciamento sono in modulo ≤ 1)
- L'inserimento di un elemento nell'AVL (ovvero, l'aggiunta di una foglia a un albero bilanciato) può provocare solo il sottocaso (i) (perché altrimenti l'AVL era già sbilanciato!)
- Invece, la cancellazione di un elemento dall'AVL (che necessariamente fa diminuire l'altezza di qualche sottoalbero) può provocare entrambi i casi (ad esempio, se cancellando un elemento ho abbassato l'altezza di T_3)
- Nel caso (i), dopo la rotazione, l'albero diminuisce la sua altezza di uno

Caso SD

[β(v)=+2, altezza T_1=h]

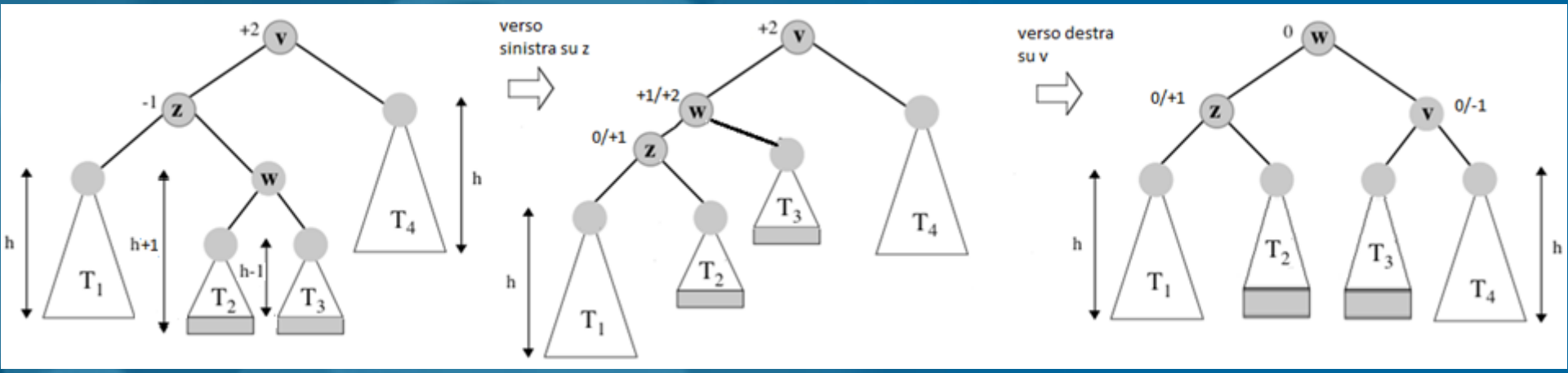
(che implica altezza T(w)=h+1)

- L'altezza di T(v) è *h*+3, l'altezza di T(z) è *h*+2, l'altezza di T_1 è *h*, l'altezza di T_4 è *h*, e l'altezza di T(w) è *h*+1 ⇒ β(v)=+2, e β(z)=-1 cioè lo sbilanciamento è provocato dal sottoalbero destro di z



- Applicare due rotazioni semplici: una verso sinistra sul figlio sinistro del nodo critico (nodo z), l'altra verso destra sul nodo critico (nodo v)

Caso SD



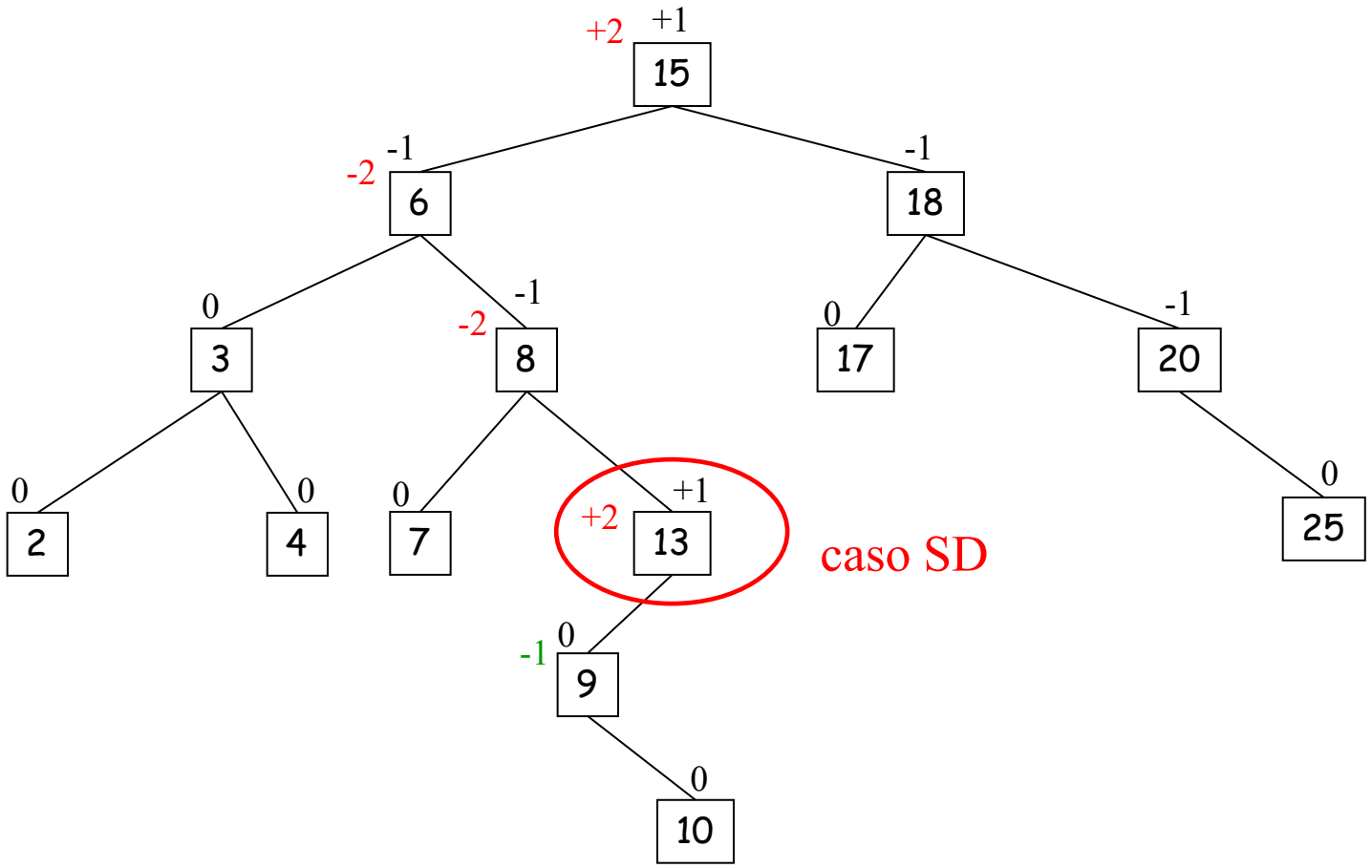
- L'altezza dell'albero dopo la rotazione passa da h+3 a h+2, poiché T_2 e T_3 sono alti al più h, e il fattore di bilanciamento di w diventa 0, mentre i fattori di bilanciamento di z e v sono 0 oppure ±1.
- Il caso SD può essere provocato sia da inserimenti (in T_2 o T_3), sia da cancellazioni che abbassano di 1 l'altezza di T_4.

insert(elem e, chiave k)

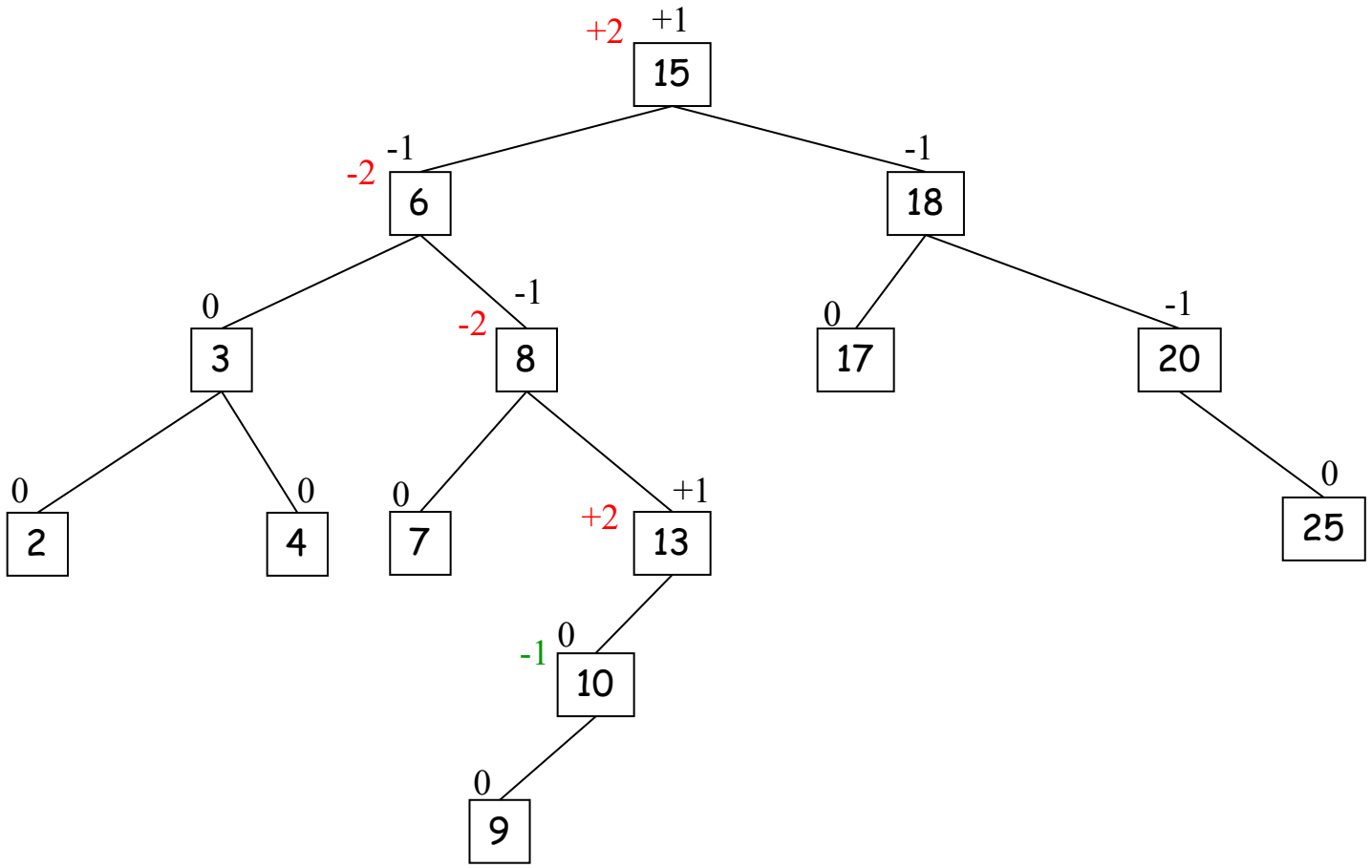
1. Crea un nuovo nodo u con elem=e e chiave=k
2. Inserisci u come in un BST
3. Ricalcola i fattori di bilanciamento dei nodi nel cammino dalla radice a u: sia v il più profondo nodo con fattore di bilanciamento pari a ±2 (nodo critico)
4. Esegui una rotazione opportuna su v

Oss.: un solo ribilanciamento è sufficiente, poiché l'altezza dell'albero coinvolto diminuisce di 1 (sottocaso (i) del caso SS o DD, o casi SD o DS), e quindi torna ad essere uguale all'altezza che aveva prima dell'inserimento

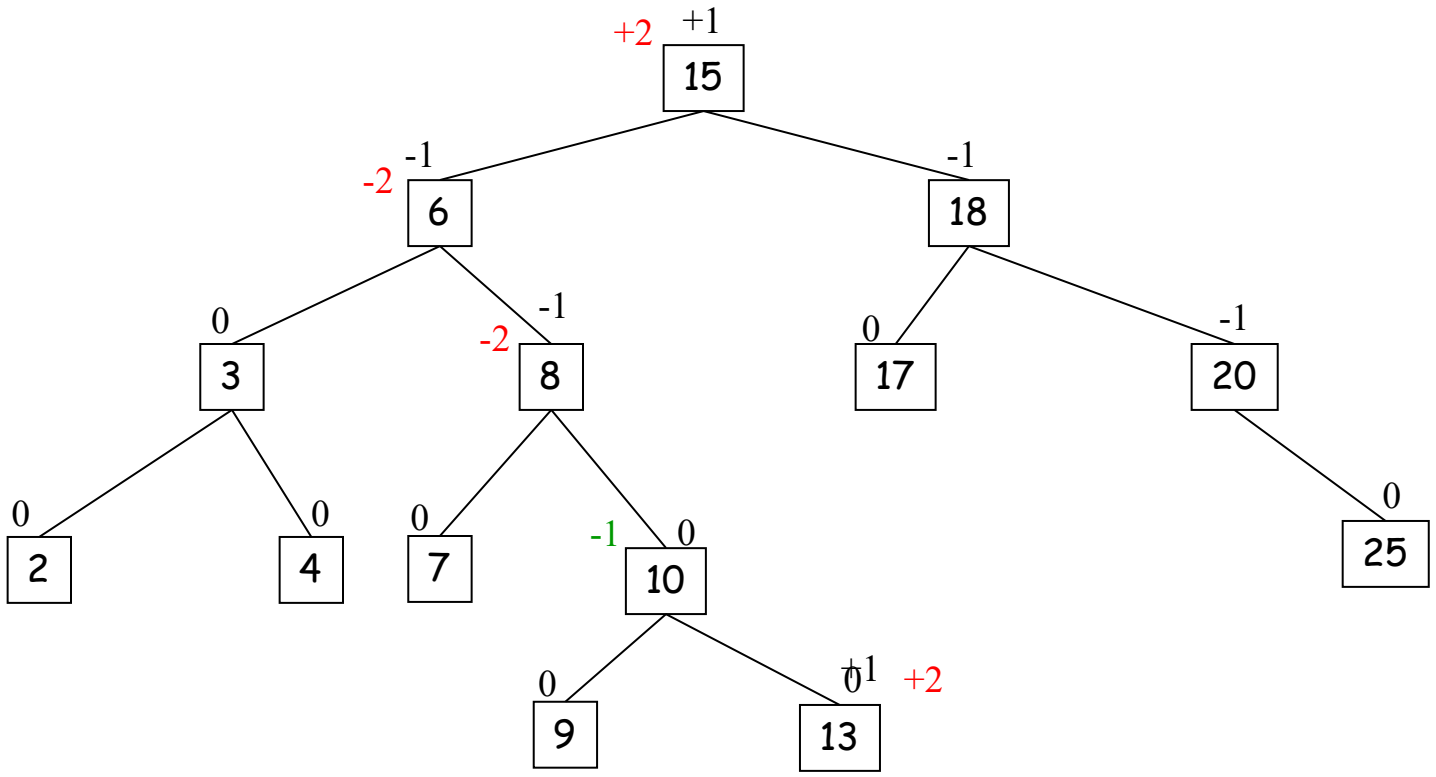
insert (10,e)



insert (10,e)



Esempio: insert (10,e)

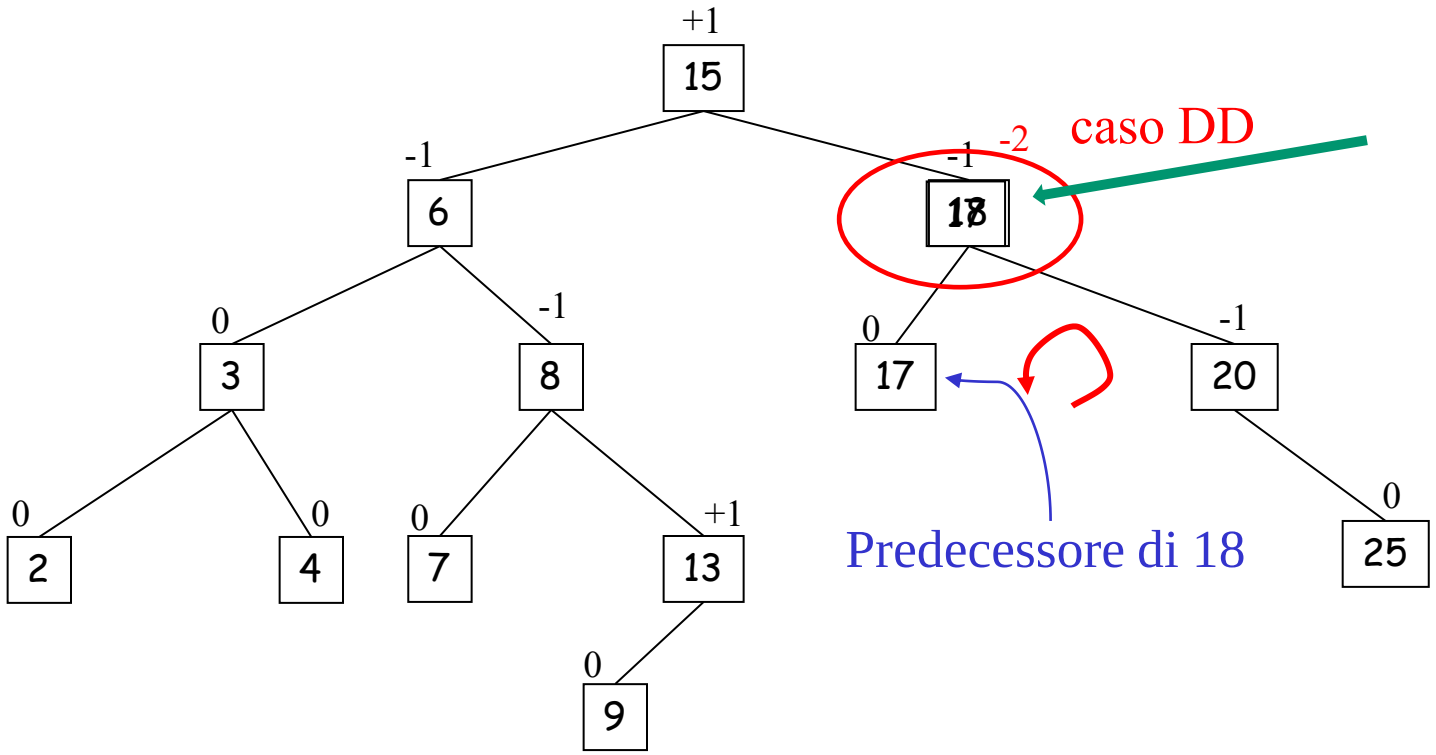


delete(elem e)

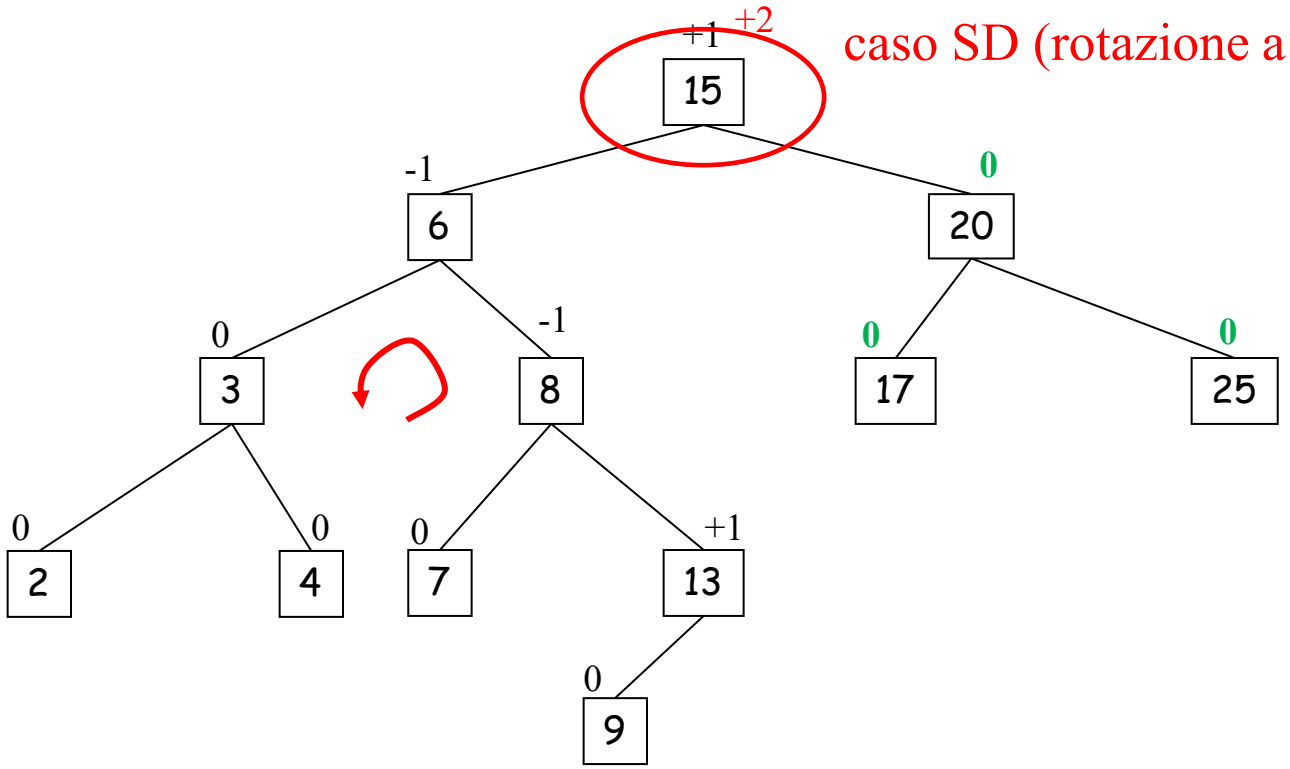
1. Cancella il nodo come in un BST
2. Ricalcola il fattore di bilanciamento del padre del nodo eliminato fisicamente (che potrebbe essere diverso dal nodo contenente e), ed esegui l'opportuna rotazione semplice o doppia ove necessario
3. Ripeti questo passo, sino ad arrivare eventualmente alla radice dell'AVL:
   - Se l'altezza del sottoalbero appena ribilanciato è uguale a quella che aveva prima della cancellazione, termina. Invece, se tale altezza è diminuita, risali verso l'alto (cioè vai nel padre del sottoalbero appena ribilanciato), calcola il fattore di bilanciamento, e applica l'opportuno ribilanciamento.

Oss.: potrebbero essere necessarie O(log n) rotazioni: infatti eventuali diminuzioni di altezza indotte dalle rotazioni possono propagare lo sbilanciamento verso l'alto nell'albero (l'altezza del sottoalbero in cui è avvenuta la rotazione diminuisce di 1 rispetto a quella che aveva prima della cancellazione)

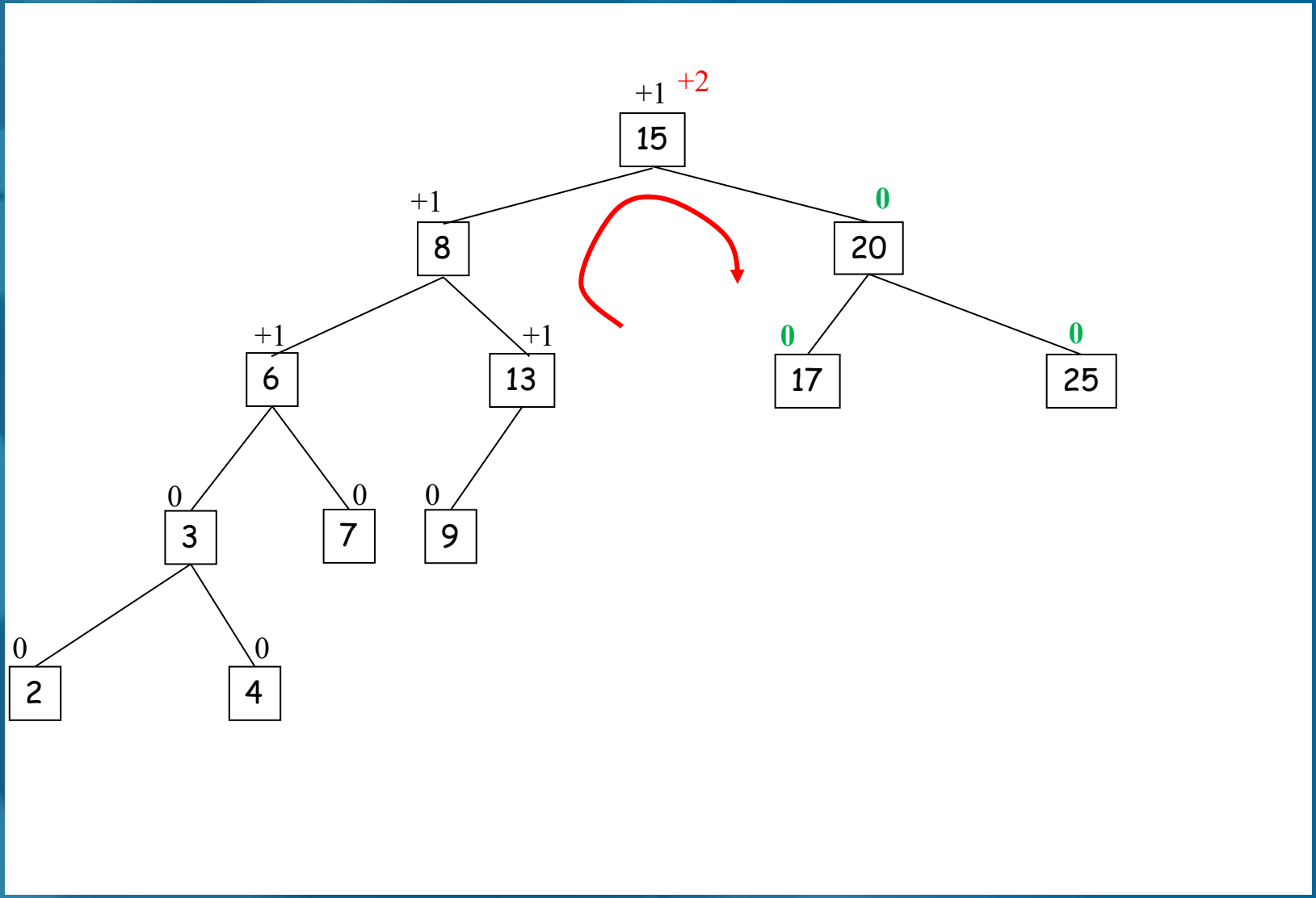
Esempio: delete (18)



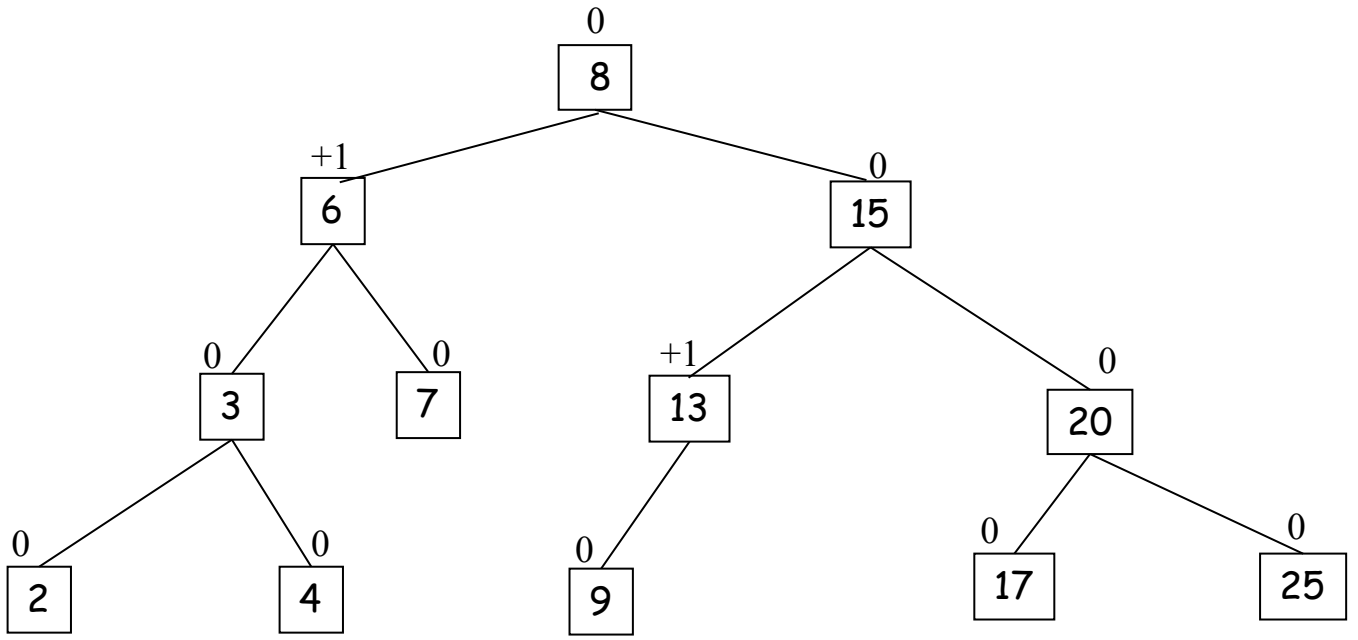
Ribilanciamento DD e aggiornamento del fattore di bilanciamento del padre del sottoalbero ruotato



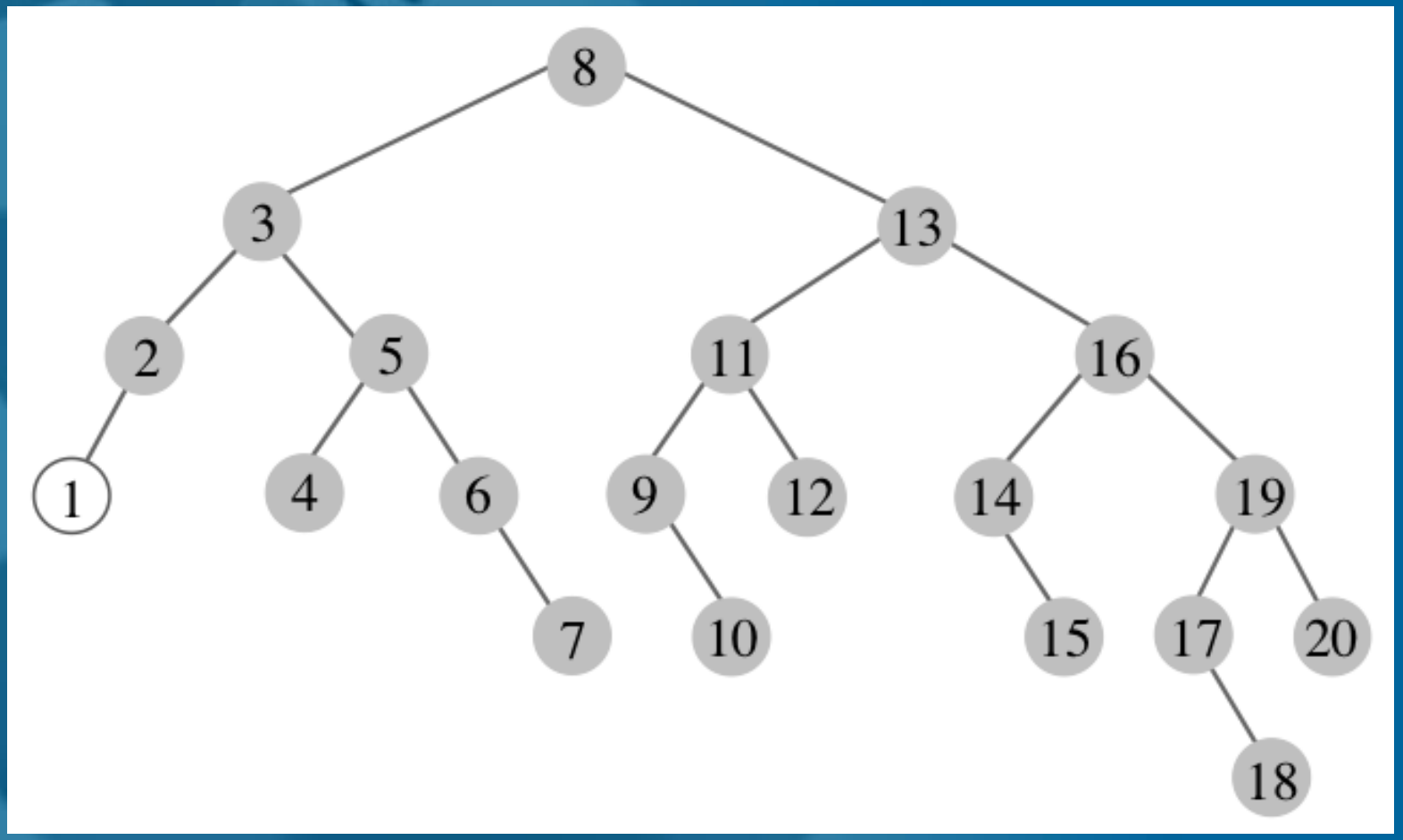
Ribilanciamento DD e aggiornamento del fattore di bilanciamento del padre del sottoalbero ruotato



Albero ribilanciato



Cancellazione con rotazioni a cascata



Costo delle operazioni

- Tutte le operazioni hanno costo O(log n) poiché l'altezza dell'albero è O(log n) e ciascuna rotazione richiede solo tempo costante

Classe AlberoAVL

classe `AlberoAVL` **estende** `AlberoBinarioDiRicerca`:

dati: $S(n) = O(n)$

albero binario di ricerca T ereditato, più il fattore di bilanciamento di ogni nodo.

operazioni:

`search` $(chiave\ k) \rightarrow elem$ $T(n) = O(\log n)$

ereditata.

`insert`$(elem\ e, chiave\ k)$ $T(n) = O(\log n)$

chiama `insert`() ereditata, poi ricalcola i fattori di bilanciamento ed eventualmente ribilancia tramite $O(1)$ rotazioni.

`delete`$(elem\ e)$ $T(n) = O(\log n)$

chiama `delete`() ereditata, poi ricalcola i fattori di bilanciamento ed eventualmente ribilancia tramite $O(\log n)$ rotazioni.

…qualche dettaglio importante.

Nell'analisi della complessità dell'operazione di insert/delete abbiamo implicitamente usato le seguenti tre proprietà:

(i) dato un nodo *v*, è possibile conoscere $\beta(v)$ in tempo $O(1)$;

(ii) dopo aver inserito/cancellato un nodo *v* nell'albero come se fosse un semplice BST, è possibile ricalcolare i fattori di bilanciamento dei nodi lungo il cammino da *v* alla radice in tempo complessivo $O(\log n)$;

(iii) nell'eseguire le rotazioni necessarie per ribilanciare l'albero, è possibile aggiornare anche i fattori di bilanciamento dei nodi coinvolti in tempo complessivo $O(\log n)$.

Esercizio

Si mostri come è possibile arricchire le informazioni contenute nel record di ogni nodo *v* in modo da garantite le proprietà (i), (ii) e (iii).

Suggerimento: aggiungere un campo al record di ogni nodo *v* che contiene l'altezza del sottoalbero radicato in *v*.