Algoritmi e Strutture Dati

Capitolo 6
Il problema del dizionario: gli alberi AVL

Riepilogo

- **Tipo dato Dizionario:** Insieme di coppie (elemento, chiave), in cui la chiave appartiene ad un dominio totalmente ordinato, sul quale eseguire operazioni di search, insert e delete.
- Implementazioni elementari: inefficienti (costo O(n))!
- **Obiettivo**: implementazione in O(log n)
- **Albero binario di ricerca**: implementazione del dizionario mediante un albero in cui ogni nodo v contiene una coppia (elemento, chiave) del dizionario, con la proprietà (che induce un ordinamento totale) che:
  - le chiavi nel sottoalbero sinistro di v sono < chiave(v)
  - le chiavi nel sottoalbero destro di v sono > chiave(v)
- Operazioni di **search** ed **insert** sull'ABR costano O(h), ove h è l'altezza dell'albero.

Interrogazioni ausiliarie su un ABR

- `max`(nodo u) – dato un nodo u di un ABR, restituisce il nodo del sottoalbero dell'ABR radicato in u avente chiave più grande (si noti che potrebbe essere u stesso, se u non ha un sottoalbero destro)
- `min`(nodo u) – dato un nodo u di un ABR, restituisce il nodo del sottoalbero dell'ABR radicato in u avente chiave più piccola (si noti che potrebbe essere u stesso, se u non ha un sottoalbero sinistro)
- `successor`(nodo u) – dato un nodo u di un ABR, restituisce il nodo dell'ABR con chiave immediatamente più grande di quella associata ad u (o NULL se u contiene l'elemento massimo dell'ABR).
- `predecessor`(nodo u) – dato un nodo u di un ABR, restituisce il nodo dell'ABR con chiave immediatamente più piccola di quella associata ad u (o NULL se u contiene l'elemento minimo dell'ABR).

Ricerca del massimo/minimo

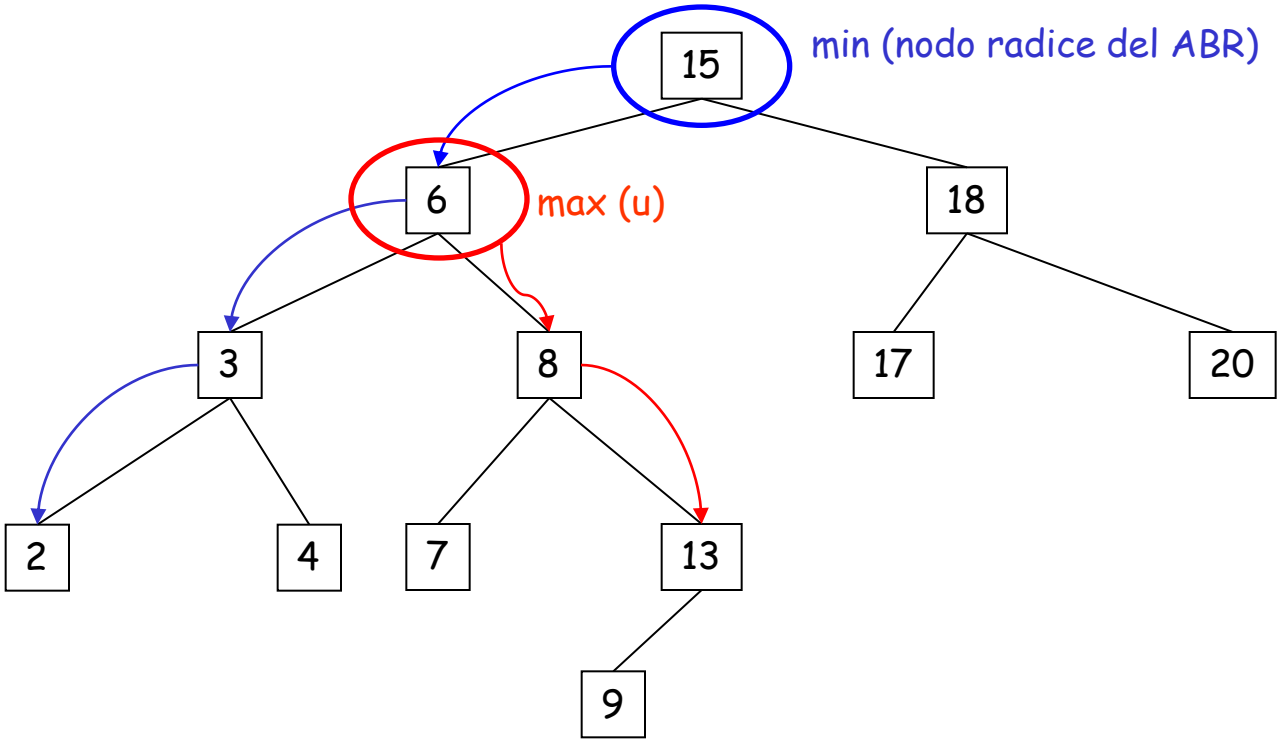
```
algoritmo max(nodo u) → nodo
1.    v ← u
2.    while ( des(v) ≠ null ) do
3.        v ← des(v)
4.    return v
```

- La procedura `min`(nodo *u*) si definisce in maniera del tutto analoga cambiando "destro" con "sinistro"

⇒ La complessità della procedura considerata è T(n) = O(h(u)), ove n è il numero di nodi dell'ABR e h(u) è l'altezza del sottoalbero radicato in u, e quindi h(u) = O(h) (h è l'altezza dell'ABR)

- Se l'argomento *nodo u* coincide con la radice dell'albero, allora `min` e `max` restituiscono il nodo con chiave minima e massima dell'intero dizionario, rispettivamente

Esempio di esecuzione



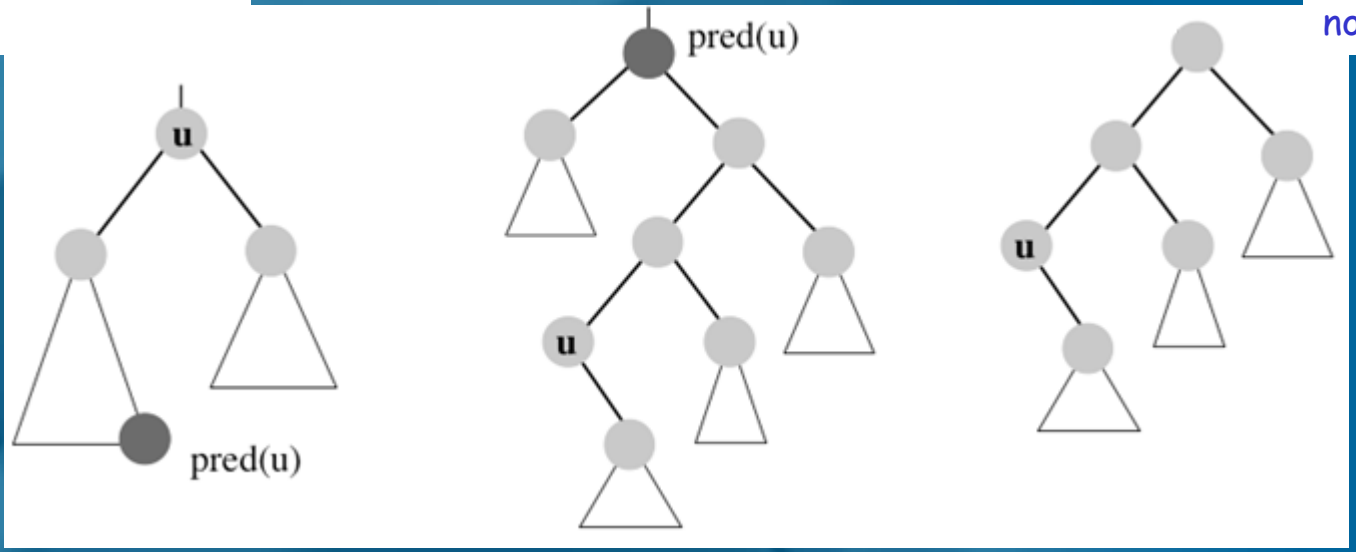
Ricerca del predecessore

Caso 1: **u** ha un sottoalbero sinistro: **max** del sottoalbero **sinistro**

Caso 2: **u** non ha un sottoalbero sinistro: Antenato più prossimo di **u** il cui sottoalbero **destro** contiene **u** (si noti infatti che **u** è il **minimo** tra gli elementi più grandi di **pred(u)**, e quindi **u** è il successore di **pred(u)**, ovvero **pred(u)** è il predecessore di **u**)

Complessità: O(h)

Caso 3: **u** è il minimo dell'albero, e quindi non ha predecessore



Ricerca del predecessore

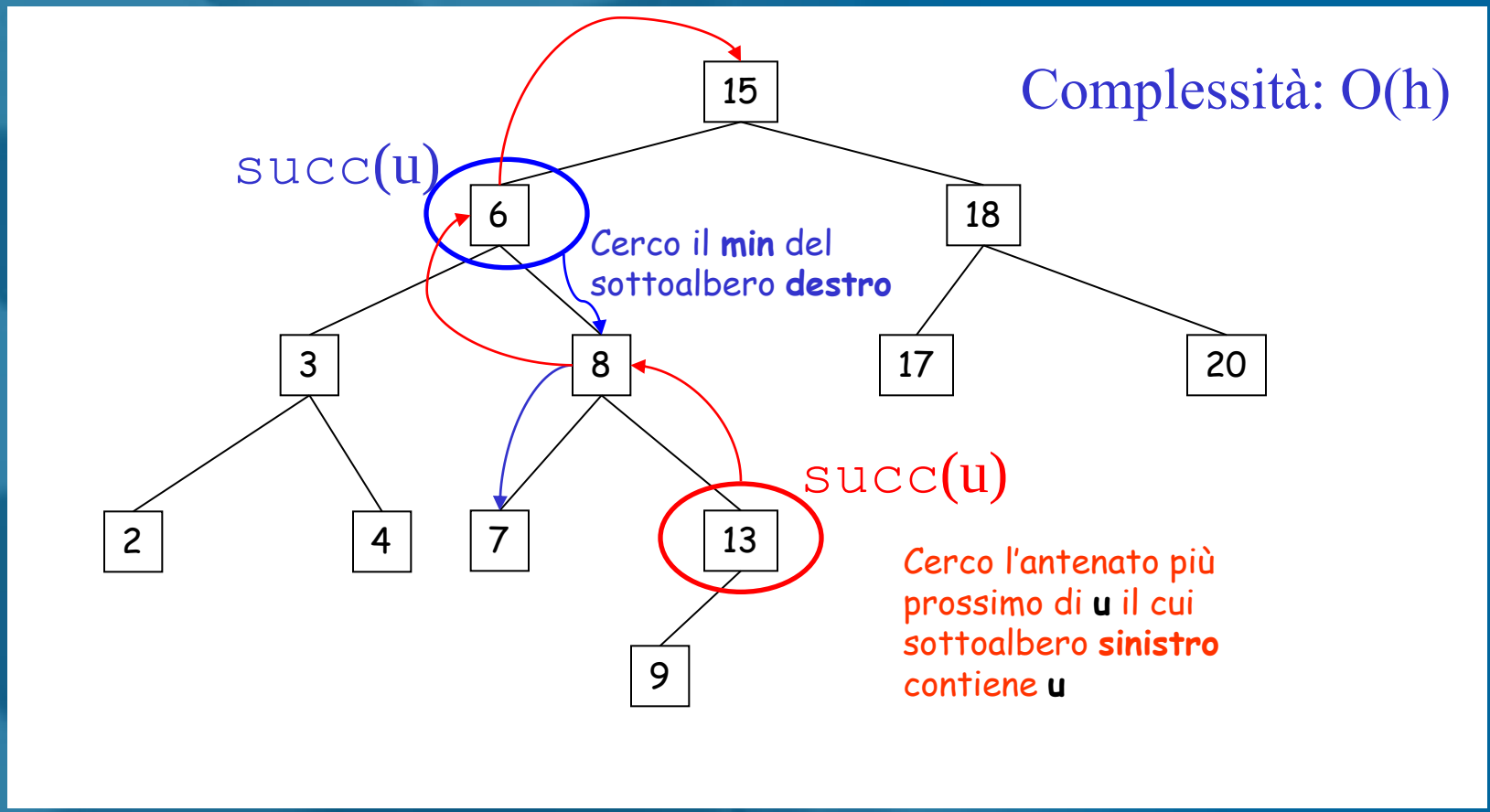
algoritmo `pred`$(nodo\ u) \rightarrow nodo$

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** $parent(u)$

Complessità: O(h)

Ricerca del successore

La ricerca del **successore** di un nodo è simmetrica: cambio "sinistro" con "destro" e "max" con "min"

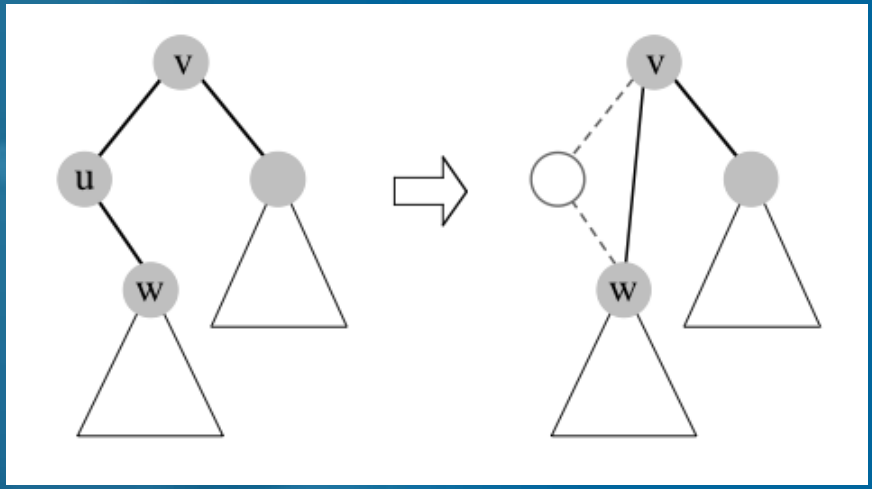


delete(elem e)

Sia u il nodo contenente l'elemento e da cancellare; ci sono 3 possibilità:

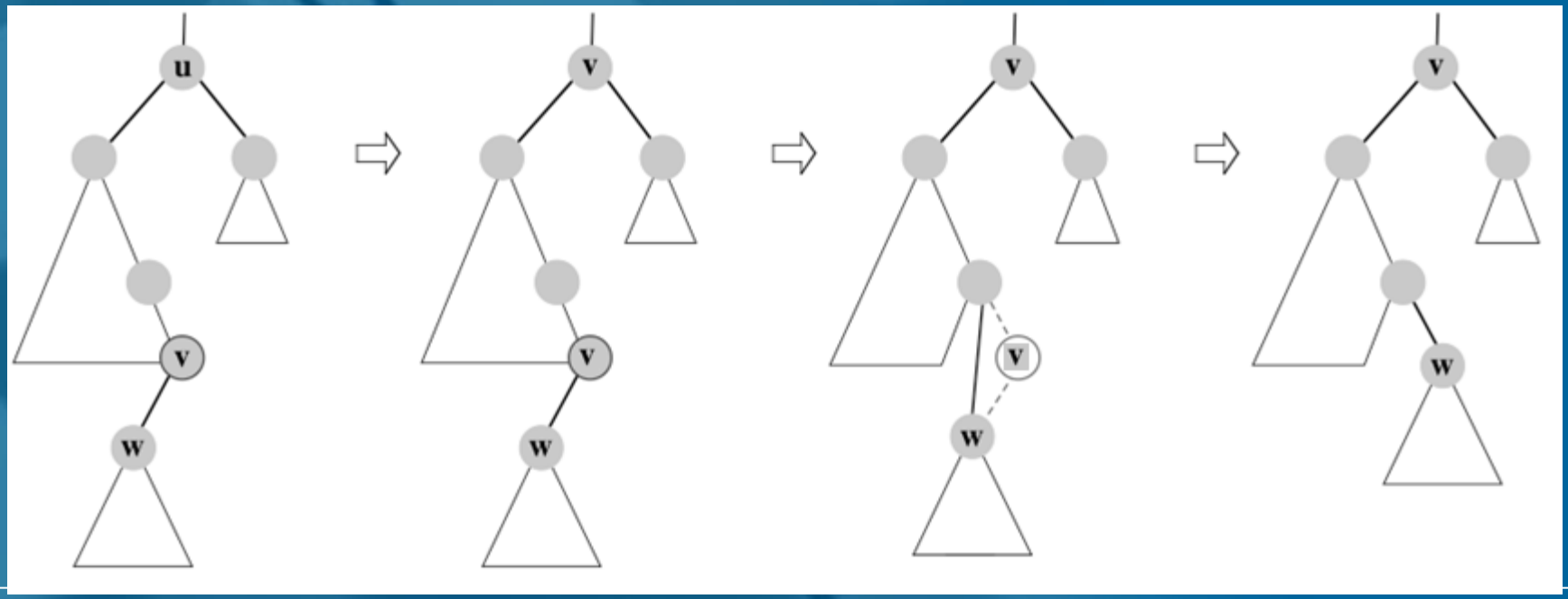
1) u è una foglia: rimuovila

2) u ha un solo figlio: aggiralo



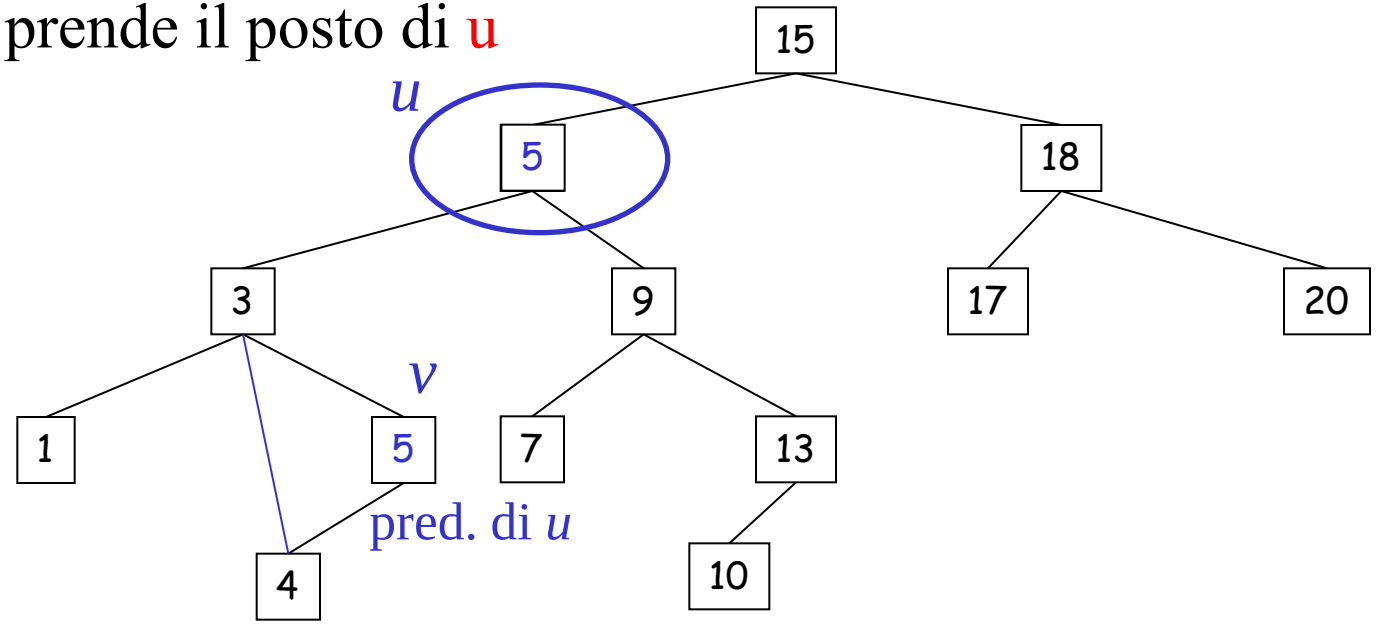
delete(elem e)

3) u ha due figli: sostituiscilo con il predecessore, e rimuovi fisicamente il predecessore; tale predecessore sarà il massimo del sottoalbero sinistro (caso 1 dell'algoritmo `pred`), in quanto u ha un sottoalbero sinistro; quindi, tale predecessore deve avere al più un solo figlio (non può avere il figlio destro!), e ricadremo quindi in uno dei 2 casi precedenti. Si noti che si può analogamente usare allo stesso scopo il successore di u. Vediamo un esempio in cui si ricade nel caso 2):



delete (*u*)

v prende il posto di u



Infine v viene aggirato

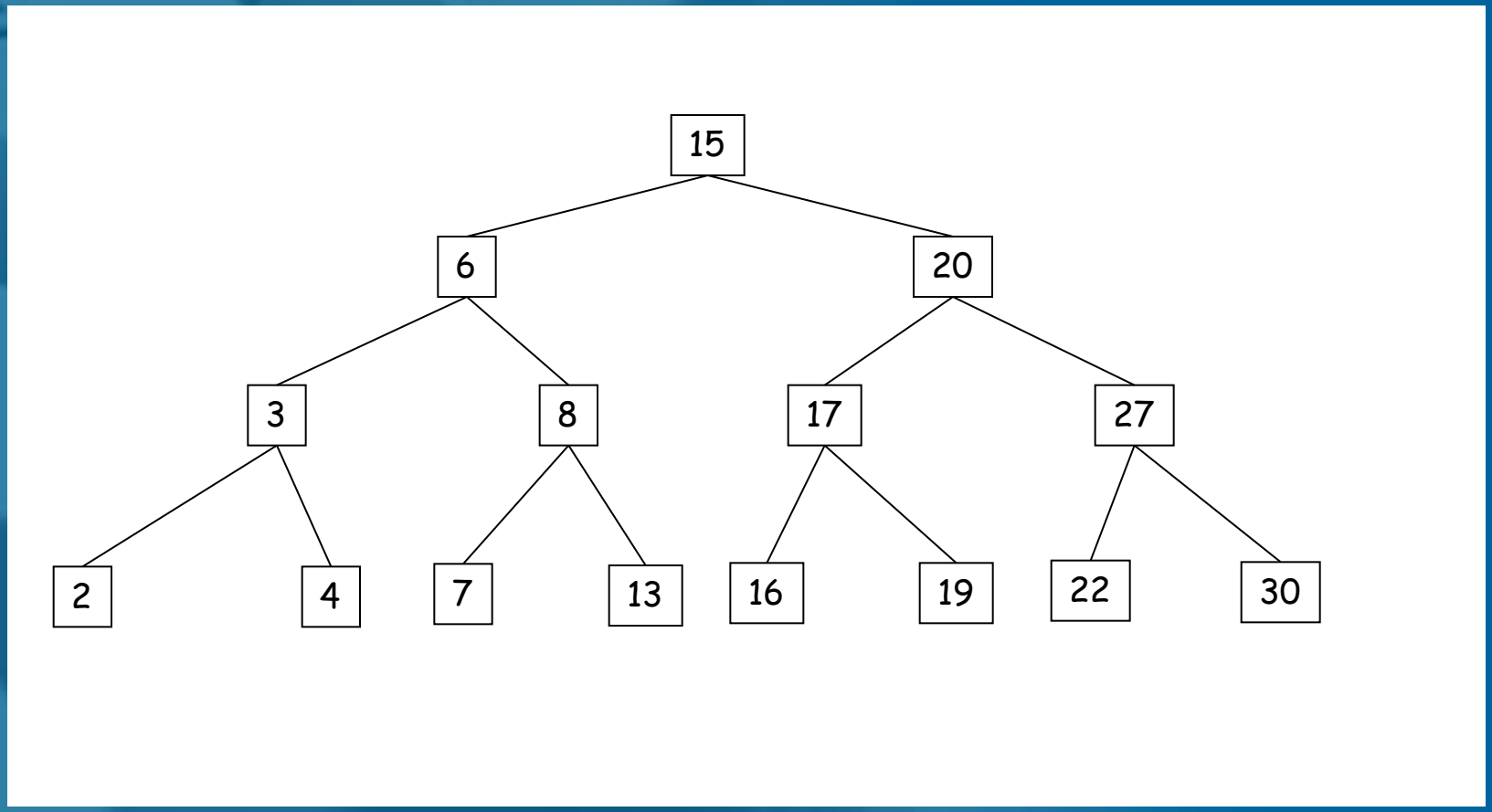
Costo dell'operazione di cancellazione

- Nei casi 1) e 2) $T(n)=O(1)$, mentre nel caso 3) $T(n)=O(h(u))=O(h)$
- Ricapitolando, le operazioni di ricerca, inserimento e cancellazione hanno costo $O(h)$ dove h è l'altezza dell'albero

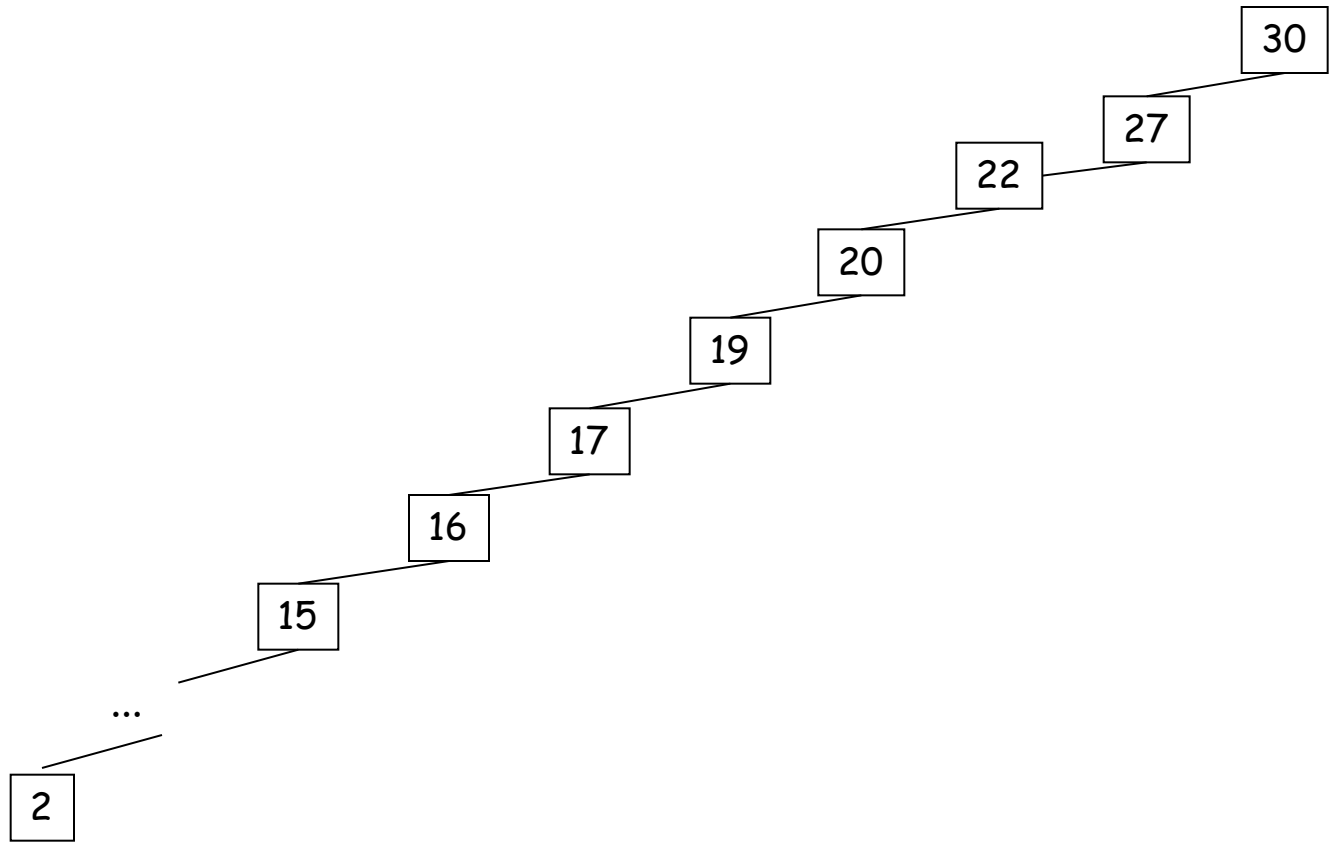
☺ Per alberi molto "bilanciati", $h= \Theta(\log n)$

☹ …ma per alberi molto "sbilanciati", $h=\Theta(n)$

Un albero binario di ricerca molto “bilanciato”…



Un albero binario di ricerca molto “sbilanciato”…



Analisi critica degli ABR

- Le operazioni di inserimento e cancellazione descritte possono "linearizzare" un ABR.

- Es. - Supponiamo di introdurre un elemento con chiave minore della chiave minima dell'ABR, poi un altro elemento con chiave ancora minore, e cosi via …

- Dobbiamo definire un modo per **mantenere** l'albero "bilanciato" (vogliamo cioè che per ogni nodo interno, le "dimensioni" dei sottoalberi sinistro e destro associati rimangano approssimativamente uguali)

$\Rightarrow$ Innanzitutto dobbiamo formalizzare il concetto di **bilanciamento**

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

Formalizzazione del bilanciamento

Fattore di bilanciamento $\beta(v)$ di un nodo v =

altezza del sottoalbero sinistro di v –

altezza del sottoalbero destro di v

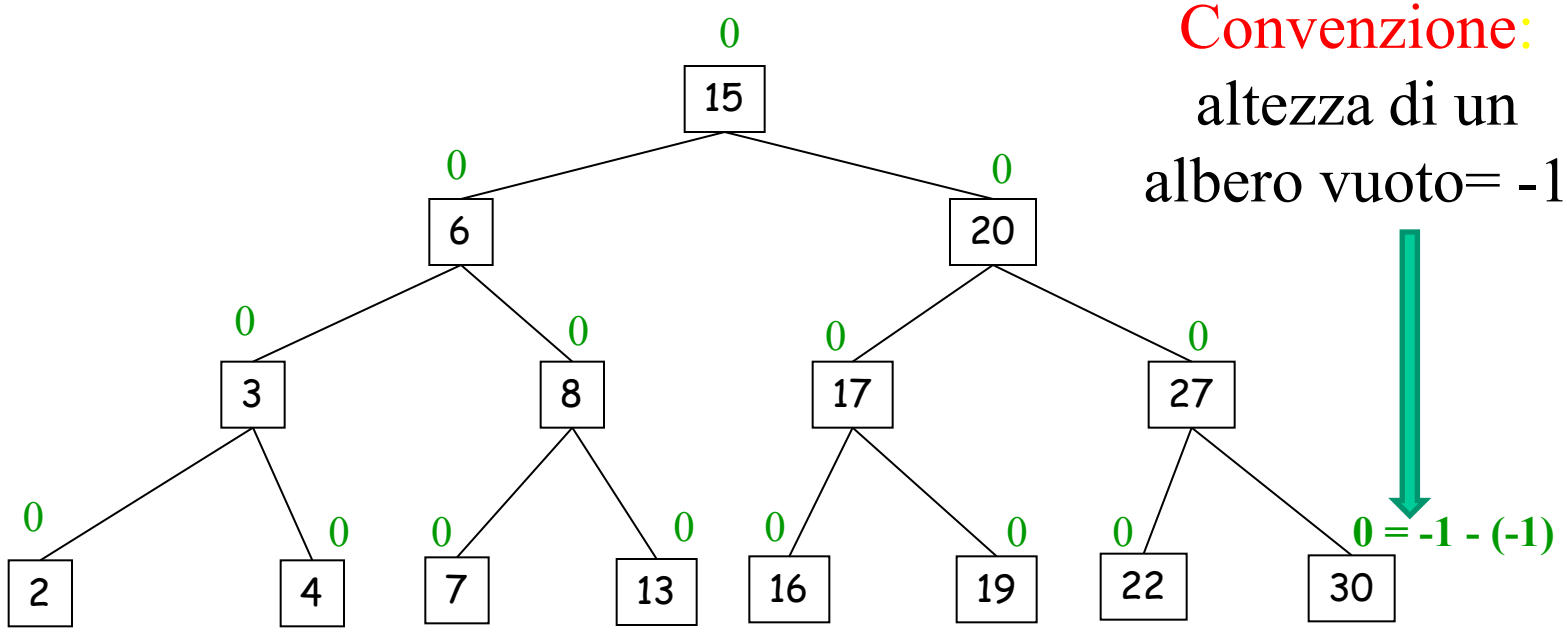
($\beta(v)$ viene mantenuto come informazione addizionale nel record associato a v, assieme alle altezze del sottoalbero sinistro e destro di v)

Def.: 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

…qualche esempio…

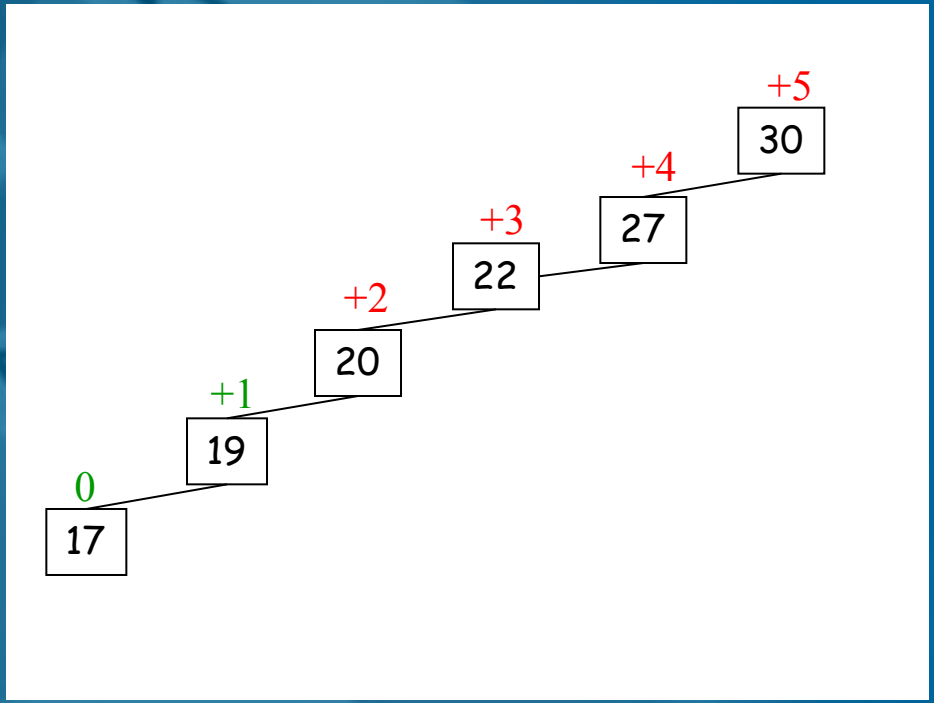
È il seguente un albero AVL?



Sì: è un ABR e tutti i nodi hanno fattore di bilanciamento = 0

…qualche esempio…

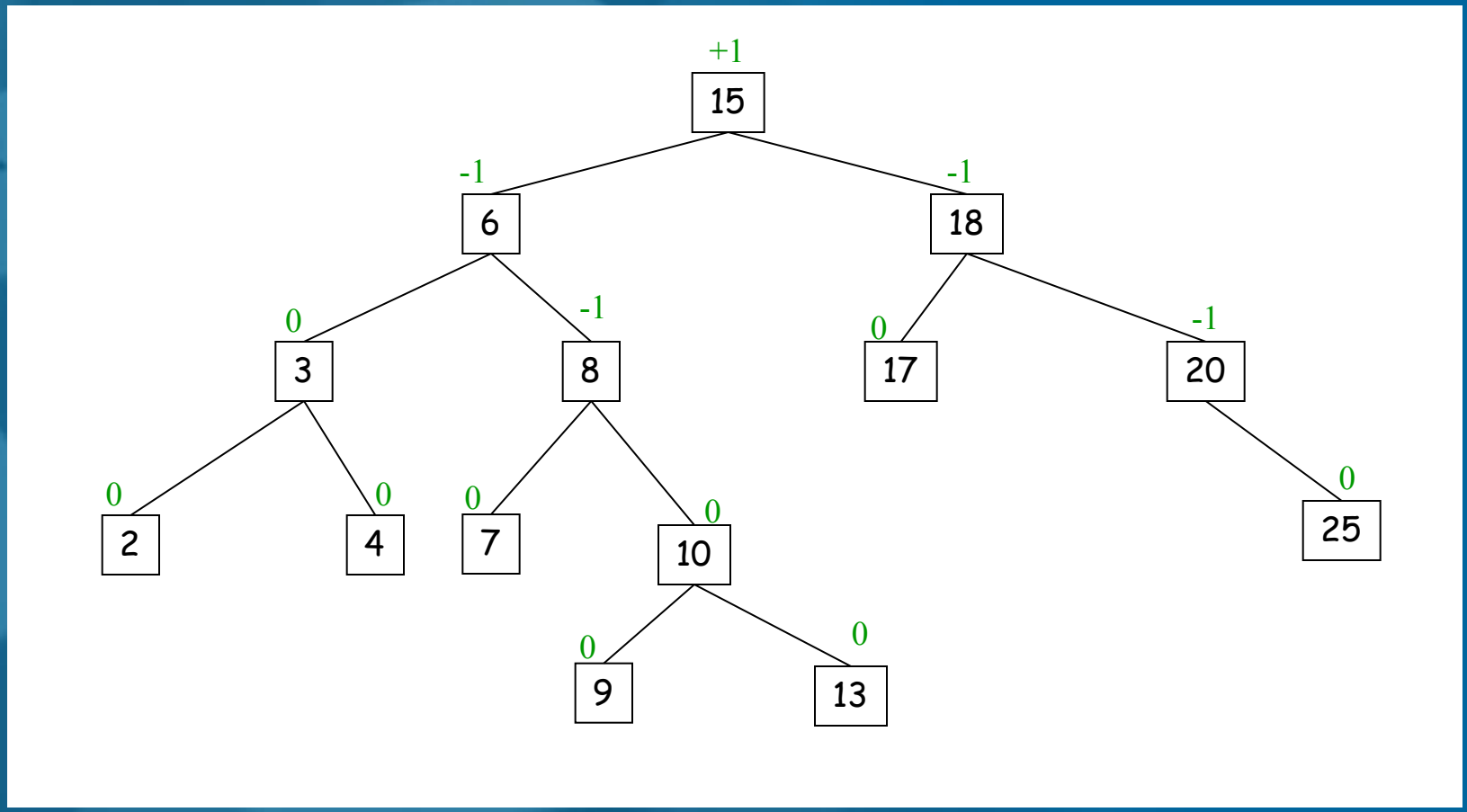
È il seguente un albero AVL?



NO! È un ABR, ma non vale la proprietà sui fattori di bilanciamento!

È il seguente un albero AVL?

…qualche esempio…



Sì: È un ABR e la proprietà sui fattori di bilanciamento è rispettata

Delimitazione superiore all'altezza di alberi AVL

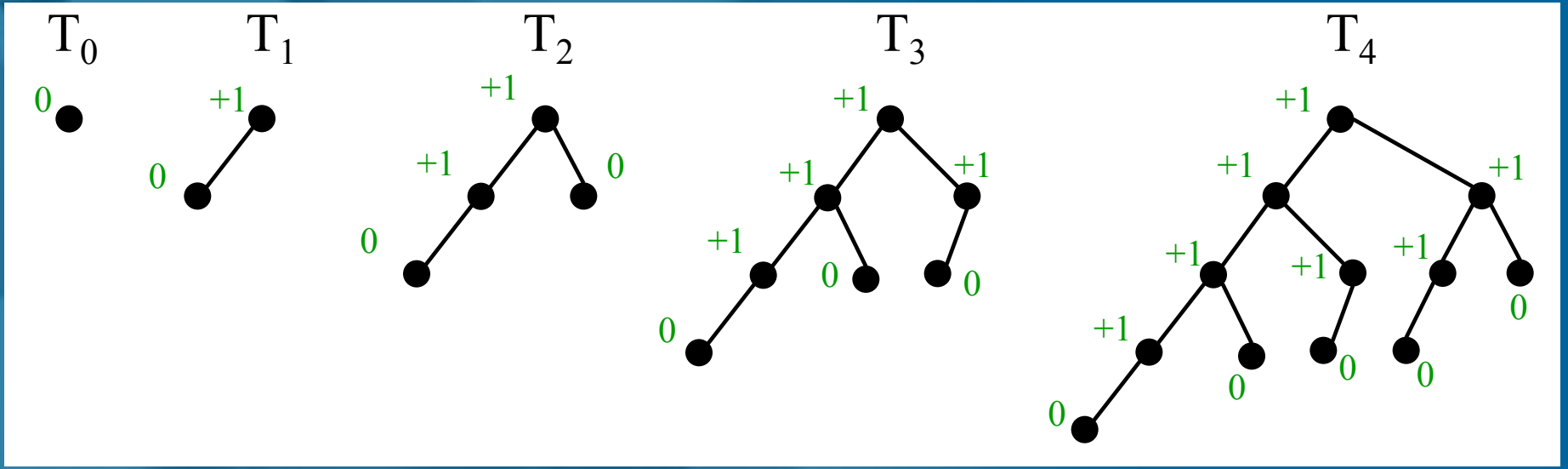
Vogliamo dimostrare che un albero AVL con n nodi ha altezza O(log n)

Idea della dimostrazione: considerare, tra tutti gli AVL di altezza h, quelli con il minimo numero di nodi n_h (che vengono detti alberi di Fibonacci), e dimostrare che $h=O(\log n_h)$

Intuizione: se per gli alberi di Fibonacci di altezza h vale $h=O(\log n_h)$, allora per tutti gli alberi AVL di altezza h con $n \geq n_h$ nodi varrà $h=O(\log n_h)=O(\log n)$

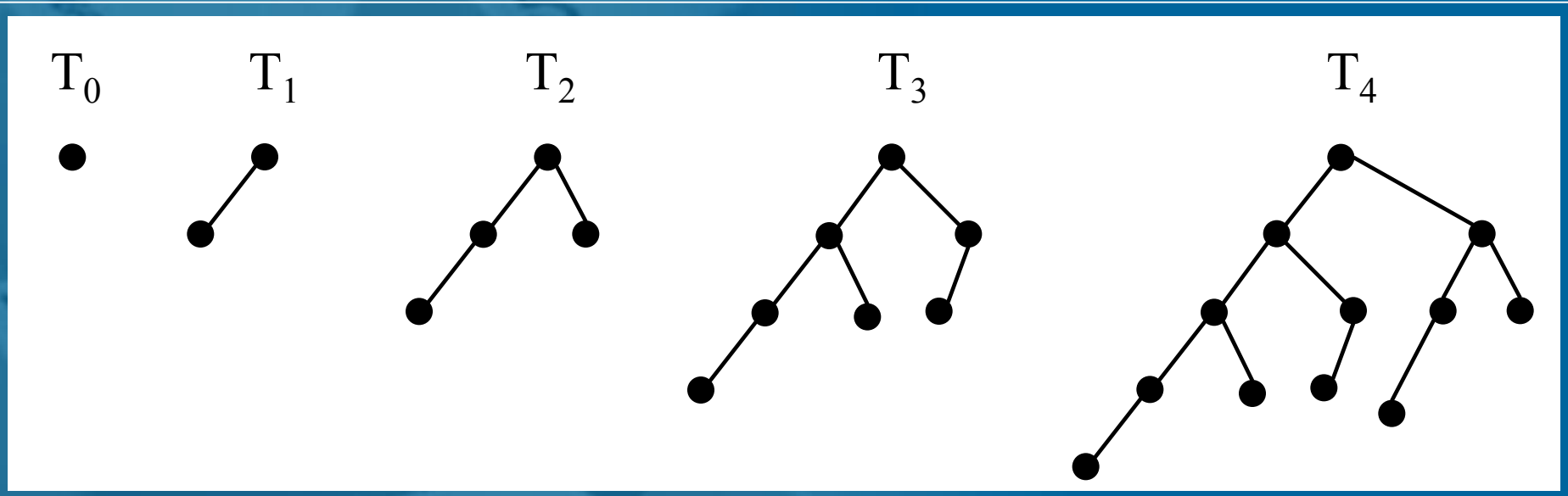
…Alberi di Fibonacci per piccoli valori di altezza…

T_h (albero di Fibonacci di altezza h): albero AVL di altezza h con il minimo numero di nodi ⇒ devo massimizzare ad 1 il fattore di bilanciamento di ogni nodo interno

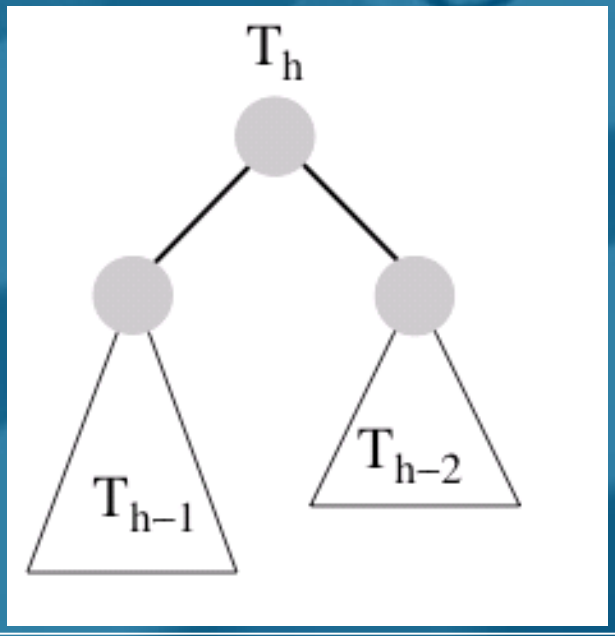


Nota: se a T_h tolgo una qualsiasi foglia (esclusa quella più in basso, che ne caratterizza l'altezza), diventa sbilanciato!

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

Dim.: Per induzione su h:

- $h=0$: $n_0 = 1$ $F_{h+3} - 1 = F_3 - 1 = 2 - 1 = 1$
- h generico:

$n_h = 1 + n_{h-1} + n_{h-2} = 1 + (F_{(h-1)+3} - 1) + (F_{(h-2)+3} - 1)$
$= 1 + (F_{h+2} - 1) + (F_{h+1} - 1) = F_{h+3} - 1.$ ■

Teorema:
Sia T un albero AVL con n nodi e di altezza h. Allora, $h=O(\log n)$.

Dim.: Si consideri l'albero di Fibonacci T_h; dal lemma

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

Ricorda che vale:
$F_h = \Theta(\phi^h)$
$\phi = 1.618\ldots$ sezione aurea

$h = \Theta(\log_\phi n_h) = \Theta(\log n_h)$,
e quindi, per l'albero T, poiché $n \geq n_h$
$h = O(\log n)$.

Delimitazione inferiore all'altezza di alberi AVL

In linea di principio, l'altezza dell'AVL potrebbe essere $o(\log n)$. Tuttavia, ciò è falso, poiché possiamo dimostrare che in un albero AVL con n nodi e di altezza h, si ha $h = \Omega(\log n)$.

Idea della dimostrazione: considerare, tra tutti gli AVL di altezza h, quelli con il massimo numero di nodi N_h (alberi binari completi)

Ma per tali alberi, $N_h = 2^{h+1}-1$, cioè $h = \Theta(\log N_h)$, e quindi per un albero AVL di altezza h con $n \leq N_h$ nodi, varrà $h = \Theta(\log N_h) = \Omega(\log n)$

Altezza di alberi AVL

Conclusione: Poiché un albero AVL con n nodi ha altezza $h=O(\log n)$ e $h=\Omega(\log n)$, ne consegue che $h=\Theta(\log n)$.