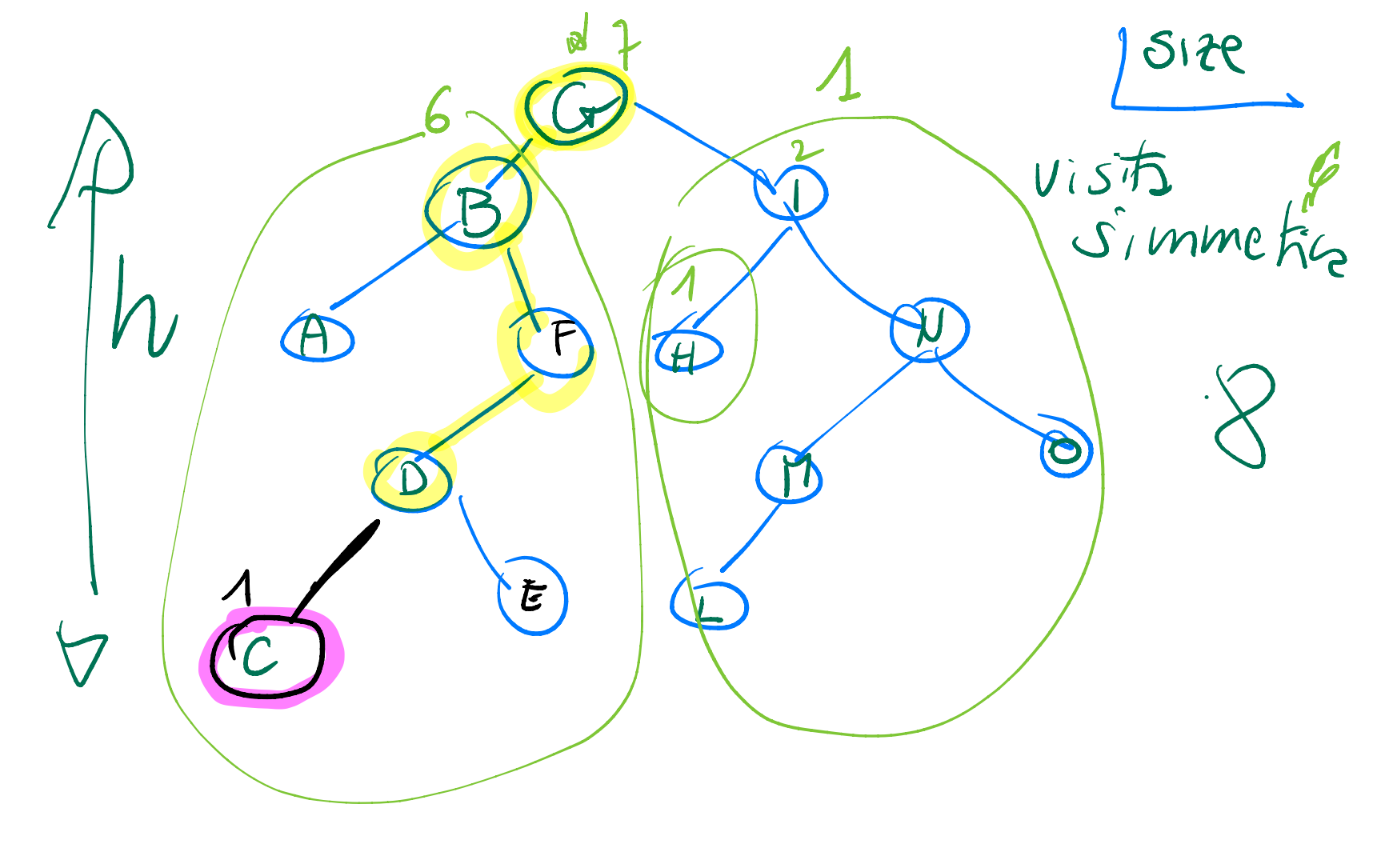

h
size
visita simmetrica
8
7
6
1
2
1
1
G
B
I
A
F
H
N
D
M
O
E
L
C

Usando gli alberi binari di ricerca, aventi n nodi e altezza h possiamo cercare, inserire e cancellare in $O(h)$ tempo invece che $O(n)$.

E' un vantaggio perché

$$\lfloor \lg(n+1) \rfloor - 1 \leq h \leq n-1$$

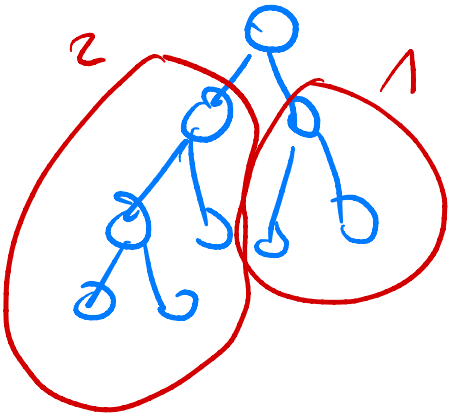
Un albero binario è BILANCIATO se $h = O(\lg n)$

Noi vediamo gli alberi AVL che hanno $h \approx 1.44 \lg n + c$

AV & L

Albero AVL = albero binario di ricerca
+
1-bilanciato

↳ $\forall$ nodo u : $|h(u_{sx}) - h(u_{dx})| \leq 1$



Q: 1-bilanciato $\overset{?}{\Rightarrow}$ $h = O(\lg n)$

Sì!

$h(u)$ = altezza di u

$h(\text{NULL}) = -1$

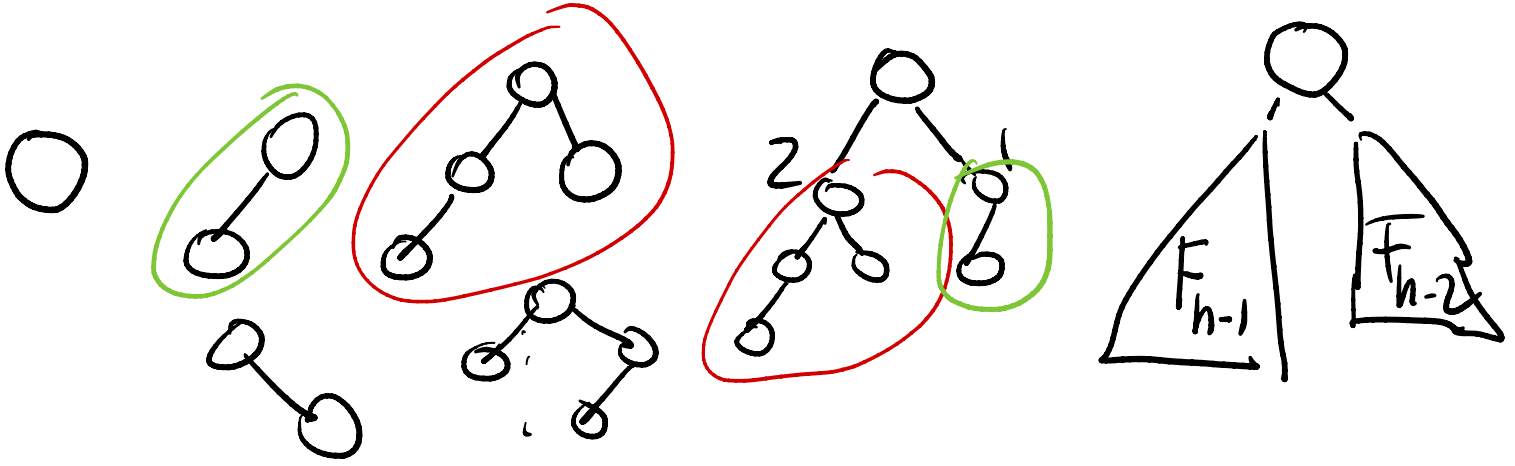
```
struct tree_node {
    int key;
    int h;
    _ _ _
    left, right }
```

Alberi di Fibonacci

Per ogni altezza h, albero binario 1-bilanciato con minimo numero n_h di nodi: F_h

h	0	1	2	3	h
n_h	1	2	4	7	$n_h = n_{h-1} + n_{h-2} + 1$

F_h



numero di nodi

altezza

h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
n_h	1	2	4	7	12	20	33	54	88	143	232	376	609	986	1596
f_h	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377

i numeri di Fibonacci

$$n_h = fib_{h+3} - 1 \quad (1)$$

$$fib_m = \frac{\Phi^m - (1-\Phi)^m}{\sqrt{5}}, \quad \Phi = \frac{1+\sqrt{5}}{2} \quad (2)$$

$(1)+(2) \Rightarrow \exists$ costante $c > 1$ t.c. $n_h \geq c^h$

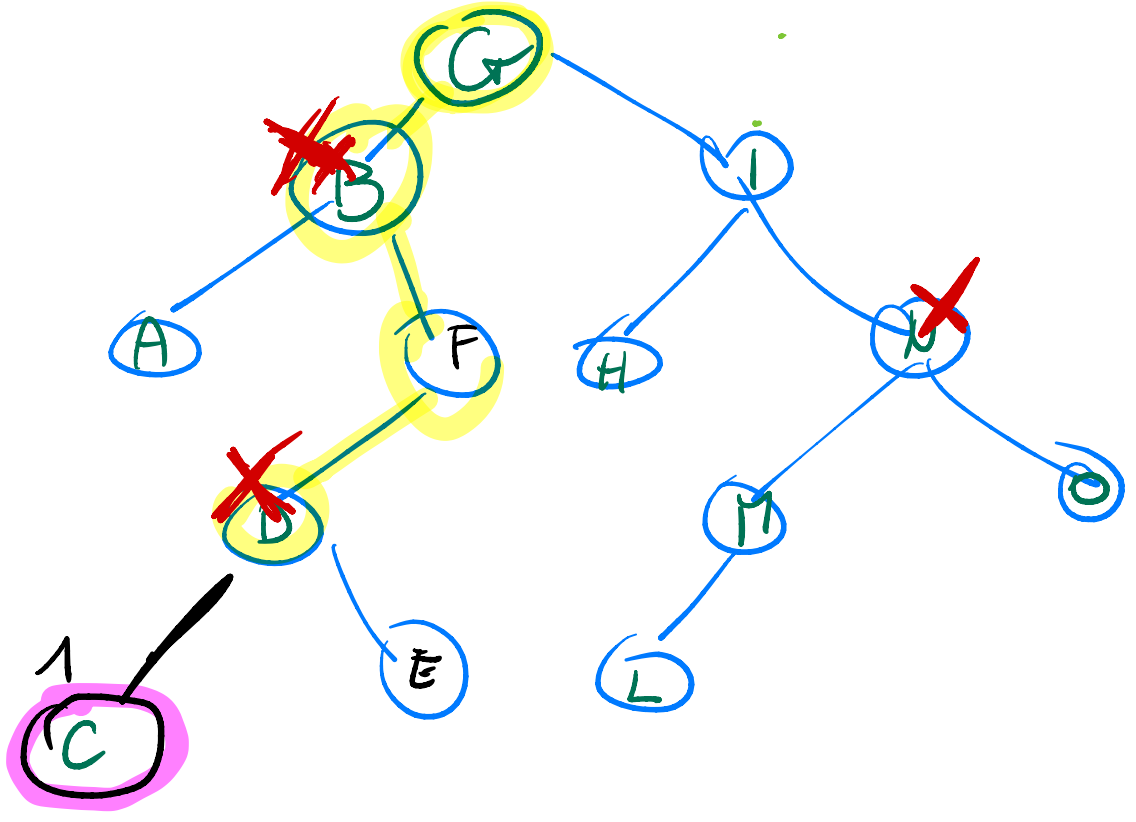
Preso un albero binario 1-bilanciato di altezza h e con n nodi, sappiamo $n \geq n_h$ (3)

$(1)+(2)+(3) \Rightarrow \exists\, c > 1$ t.c. $n \geq n_h \geq c^h$

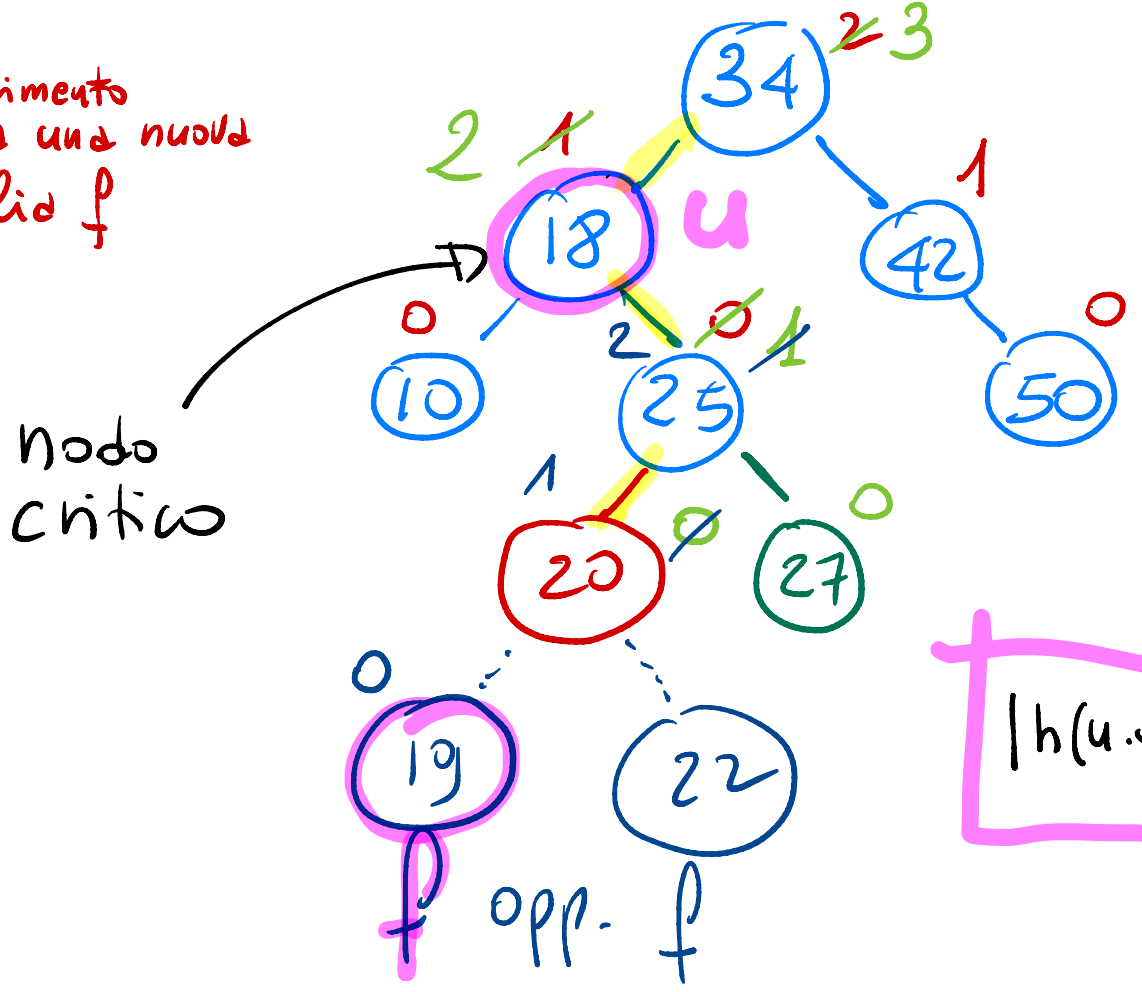
$\Rightarrow h = O(\log n)$

- RICERCA AVL = RICERCA A.B.R.
- INSERIMENTO AVL : OGGI!
- CANCELLAZIONE AVL (la facciamo logica)

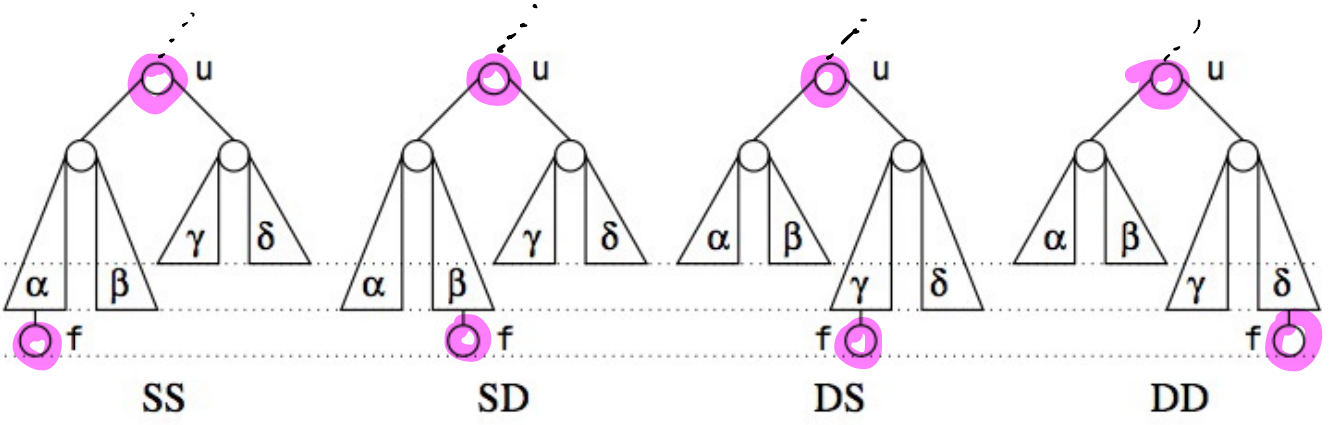
↑ il nodo viene marcato come "non valido" e l'albero ricostruito periodicamente


G
B
I
A
F
H
N
D
M
O
E
L
1
C

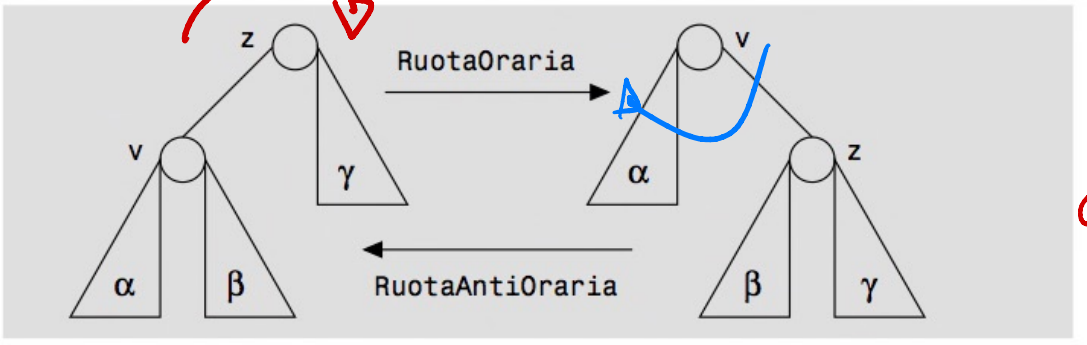
Inserimento crea una nuova foglia f



$$|h(u.sx) - h(u.dx)| = 2$$



Introduciamo le rotazioni

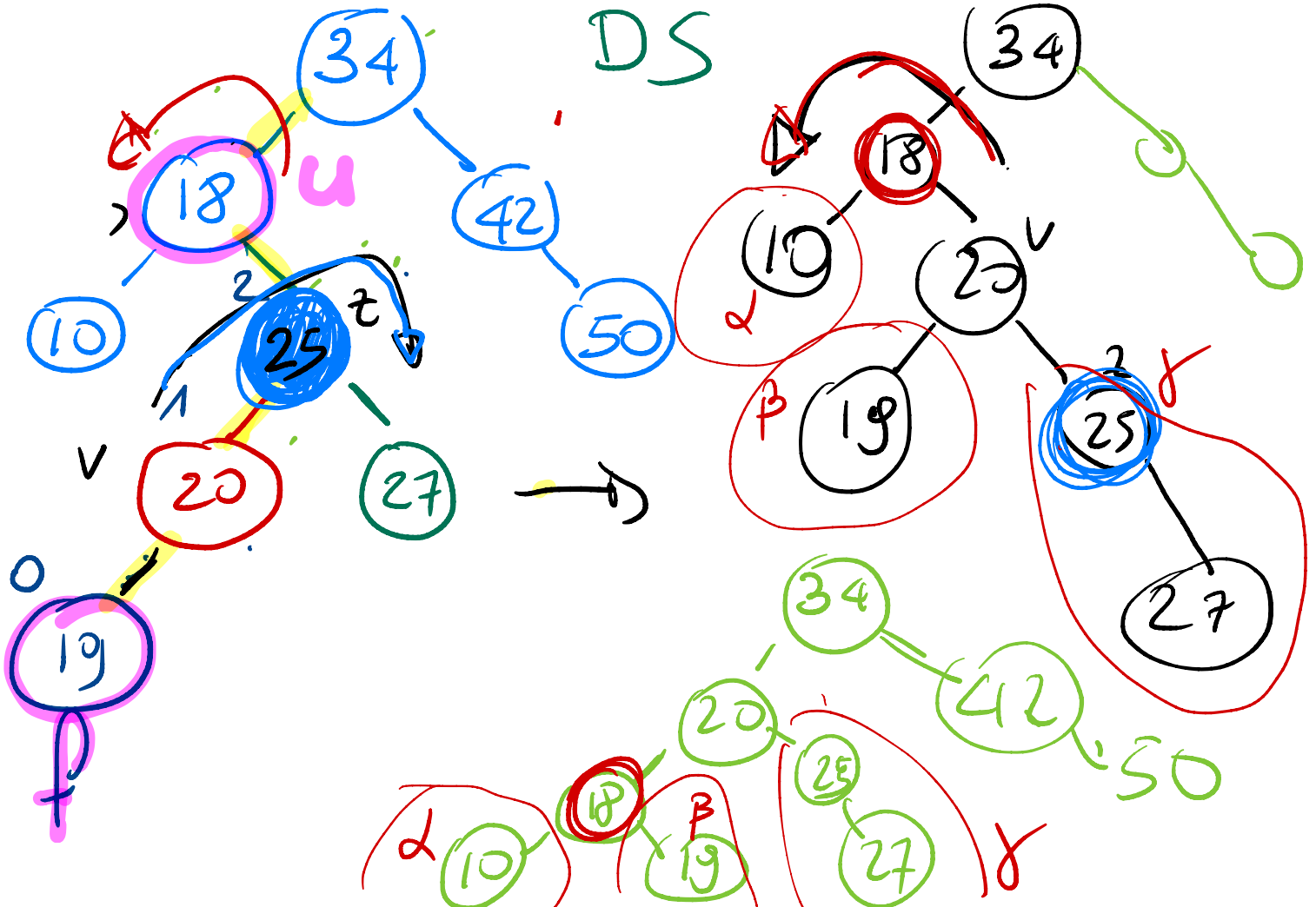


$\alpha \leq v \leq \beta \leq z \leq \gamma$

```
1  RuotaOraria( z ):
2    v = z.sx;
3    z.sx = v.dx;
4    v.dx = z;
5    z.altezza = max( Altezza(z.sx), Altezza(z.dx) ) + 1;
6    v.altezza = max( Altezza(v.sx), Altezza(v.dx) ) + 1;
7    RETURN v;
```

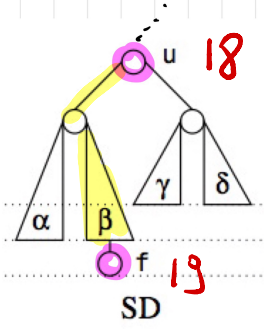
$O(1)$ tempo

```
1  RuotaAntiOraria( v ):
2    z = v.dx;
3    v.dx = z.sx;
4    z.sx = v;
5    v.altezza = max( Altezza(v.sx), Altezza(v.dx) ) + 1;
6    z.altezza = max( Altezza(z.sx), Altezza(z.dx) ) + 1;
7    RETURN z;
```

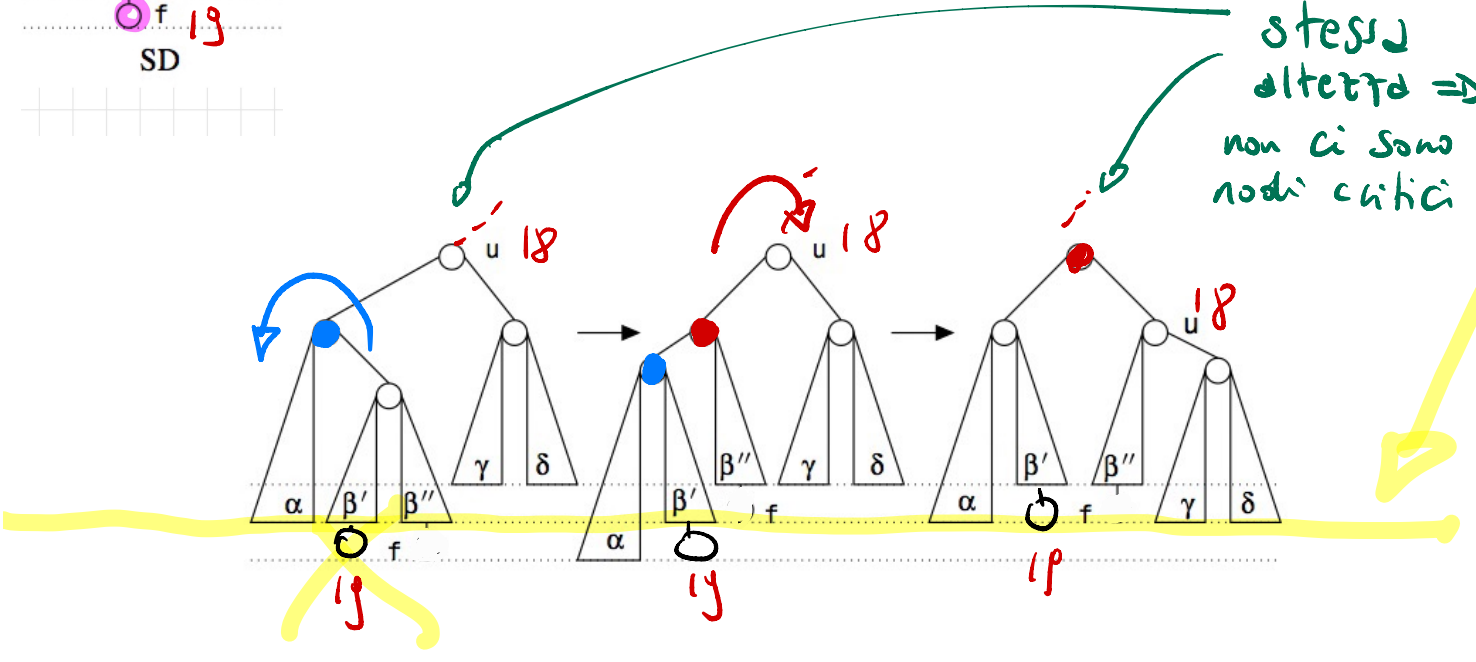

DS
34
18
42
10
25
50
20
27
19
u
x
z
v
w
34
18
10
25
19
25
27
34
20
42
50
18
25
10
19
27
α
β
γ

DS ordina

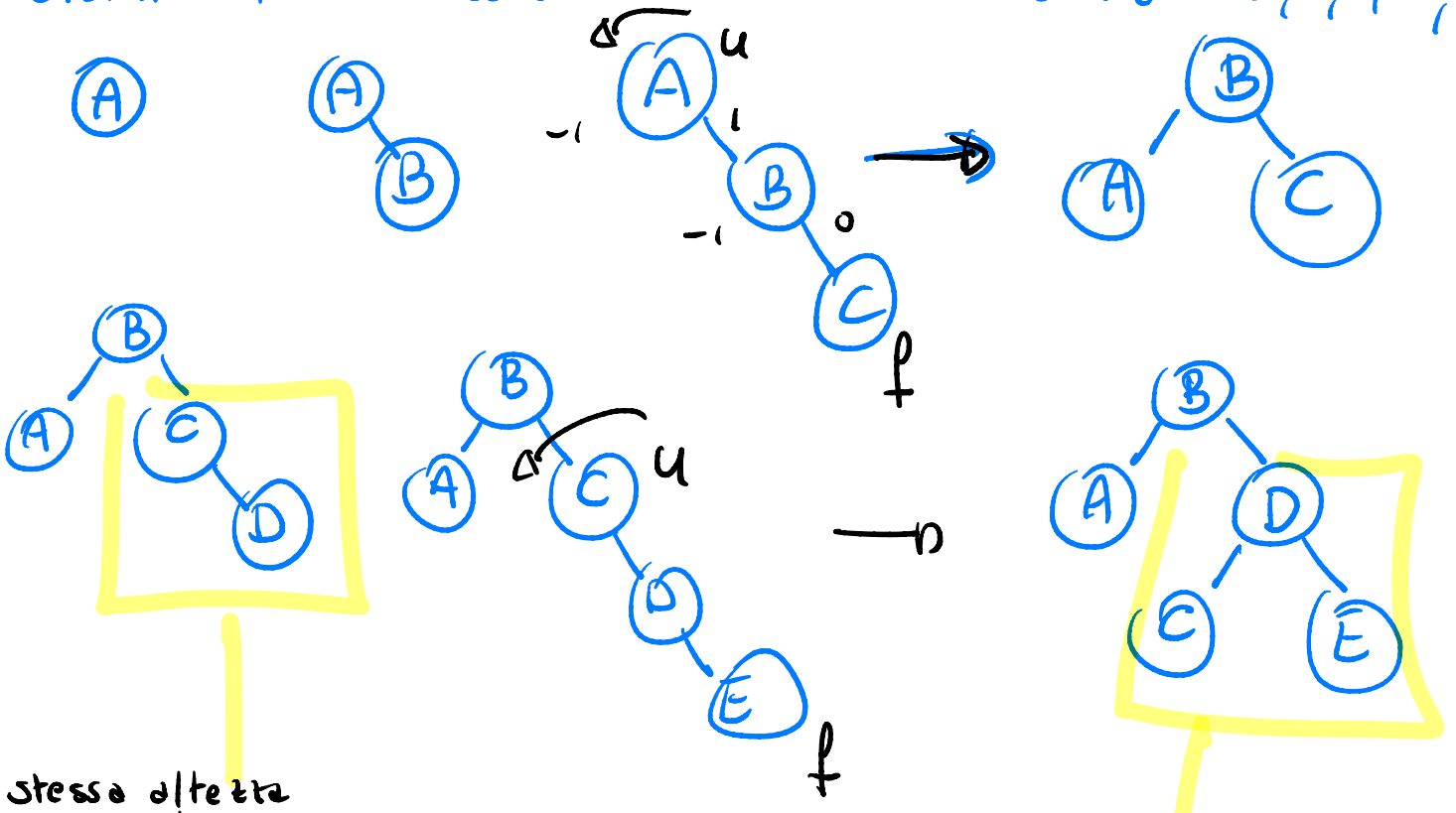
antenato (u)



stessa altezza ⇒ non ci sono più nodi critici



CRASH TEST .... inseriamo chiavi in ordine crescente A, B, C, D, E, F...



```
1  Inserisci( u, e ):
2    IF (u == null) {
3      RETURN f = NuovaFoglia( e );
4    } ELSE IF (e.chiave < u.dato.chiave) {
5    u.sx = Inserisci( u.sx, e );
6    IF (Altezza(u.sx) - Altezza(u.dx) == 2) {
7      IF (e.chiave>u.sx.dato.chiave) u.sx=RuotaAntiOraria(u.sx);
8      u = RuotaOraria( u );
9    }
10   } ELSE IF (e.chiave > u.dato.chiave) {
11   u.dx = Inserisci( u.dx, e );
12   IF (Altezza(u.dx) - Altezza(u.sx) == 2) {
13     IF (e.chiave < u.dx.dato.chiave) u.dx = RuotaOraria(u.dx);
14     u = RuotaAntiOraria( u );
15   }
16   }
17   u.altezza = max( Altezza(u.sx), Altezza(u.dx) ) + 1;
18   RETURN u;
```

già visto

NODO CRITICO?

SD?

```
1  Altezza( u ):
2    IF (u == null) {
3      RETURN -1;
4    } ELSE {
5      RETURN u.altezza;
6    }
```

```
1  NuovaFoglia( e ):
2    u = NuovoNodo();
3    u.dato = e;
4    u.altezza = 0;
5    u.sx = u.dx = null;
6    RETURN u;
```

k = e.chiave

ROTAZIONE in u

SS / SD

DD / DS

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

tempo

DIZIONARIO

U = universo delle chiavi

$n = |S|$

$S \subseteq U$ da memorizzare

- appartenenza $x \in S$?
- inserimento $S \leftarrow S \cup \{x\}$
- cancellazione $S \leftarrow S \setminus \{x\}$

C++: ordered < map / set unordered < map / set

* non include il costo di ricerca

Struttura di dati per il DIZIONARIO	appartenenza (RICERCA)	inserimento	cancellazione * (siamo già lì)
array non ordinato	$O(n)$	$O(1)$	$O(1)$
array ordinato	$O(\lg n)$	$O(n)$	$O(n)$ [$O(1)$ logica]
lista non ordinata	$O(n)$	$O(1)$	$O(n)$ | $O(1)$ (conoscendo il predec.)
lista ordinata	$O(n)$	$O(1)$ siamo già lì	↑ vedo sopra
albero binario di ricerca	$O(h)$	$O(h)$	$O(h)$ $h \le n-1$ ordine casuale ok
albero AVL come sopra ma $h = O(\lg n)$	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$ | $O(1)$ se logica
tabelle HASH (non è ordinata)	$O(1)$ medio	$O(1)$ medio	$O(1)$ medio

→ prossima lezione h = altezza