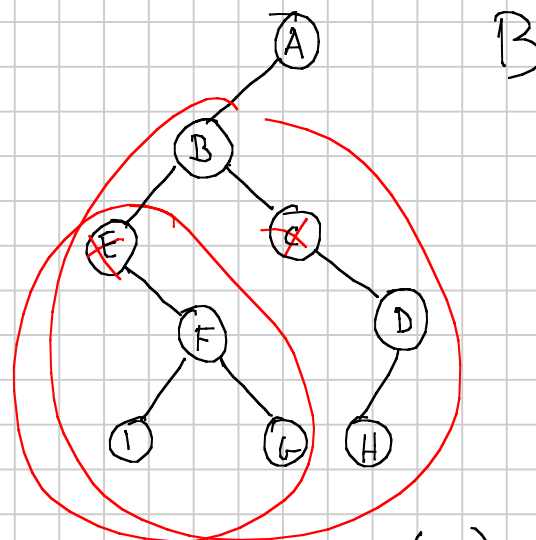
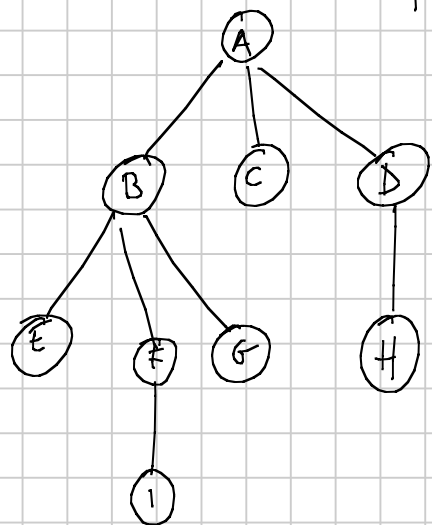


Alberi binari



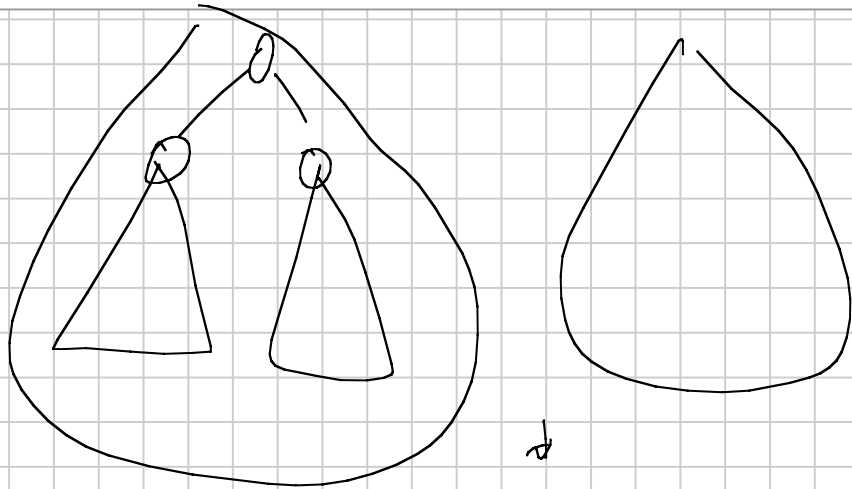
Pre-visita (T) = A B E F I G C D H

Pre-visita (B) = A B E F I G C D H

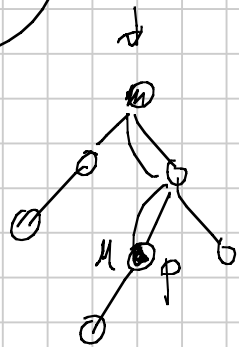
Invisita (B) = E I F G B C H D A

Post visita (T) = E I F G B C H D A

Post visita (B) = I G F E H D C B A



BOTTOM-UP
 TOP-DOWN
 PROFONDITÀ DI TUTTI I NODI
 DELL'ALBERO



$prof.(u) = 2$

$O(nh) \Rightarrow O(n)$
 meglio

calcolando le profondità durante una visita
 propagandola dalle radici verso il basso

```
Prof ( u, p ) :  
if u == NULL  
    print <  $\phi$ ,  $\phi$  >;  
else {  
    print < u.Key, p >;  
    prof ( u.lx, p+1 );  
    prof ( u.rx, p+1 );  
}
```

Stampa la profondità di
tutti i nodi dell'albero $\langle \text{key}, p \rangle$
basato Pre-Visite

$\Theta(n)$

Trovare tutti i nodi CARDINE, tali per cui
altezza = profondità.

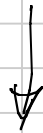
- Due
- raccogliere l'informazione dai sott alberi
 - propagare l'informazione degli esterni.

$$\Theta(n)$$

Alberi binari di ricerca
(realizzati con puntatori)



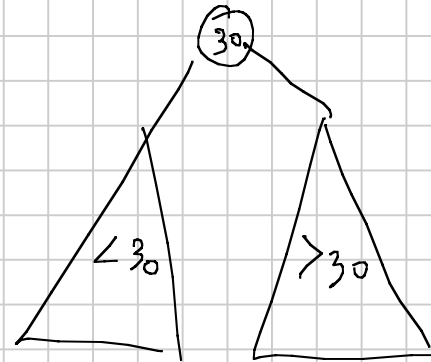
Struttura



relazioni tra
le chiavi

Alberi per realizzare dizionari

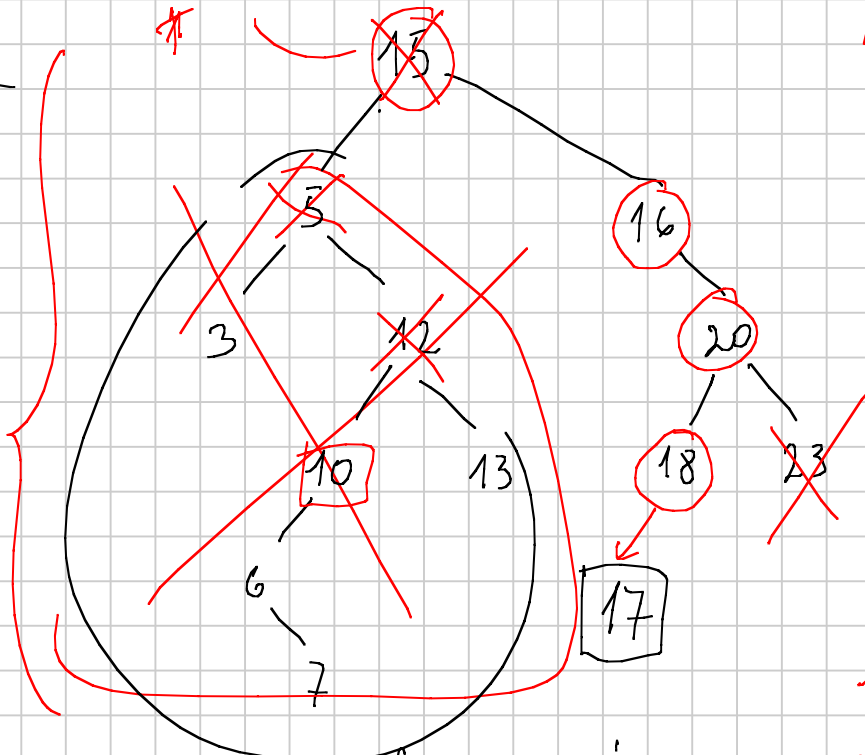
tutte le chiavi
sono distinte



proprietà valide
per tutte le chiavi
dell'albero

Alberi di Ricerca

h
altezza
dell'albero

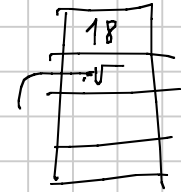


Ricerca di 17

La ricerca termina senza
successo su un puntatore = NULL

Ricerca di 10

$$O(h)$$

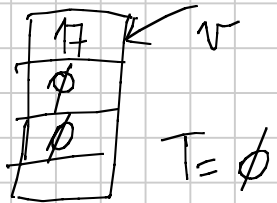


$h = O(\log n)$ per alberi di
ricerca bilanciati

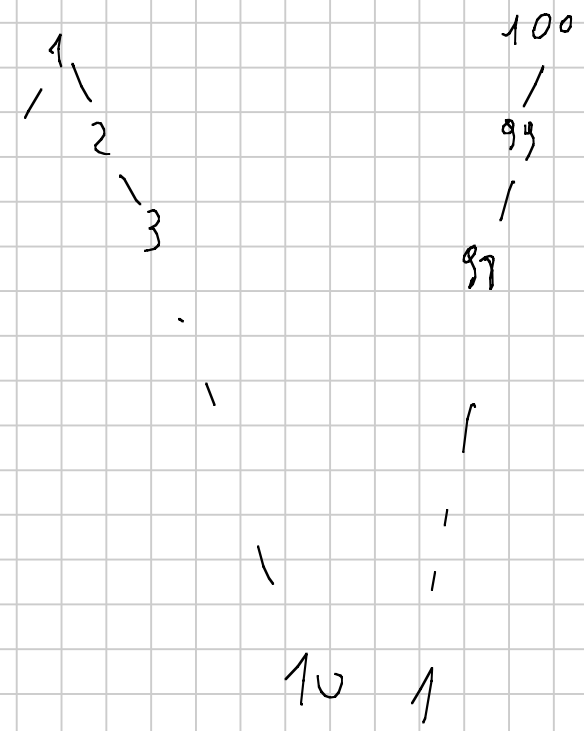
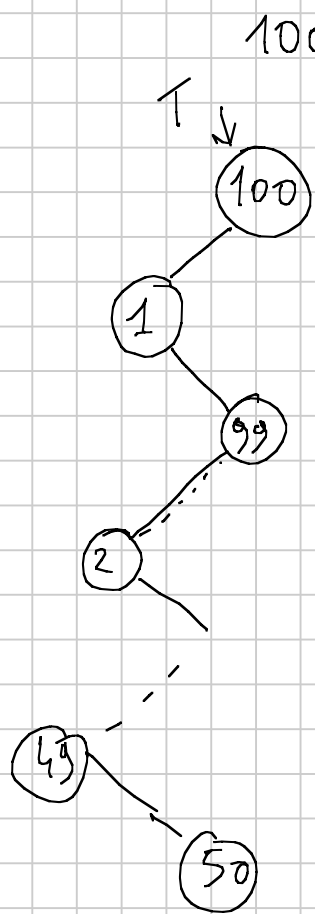
Inserzione di una nuova chiave

Inserzione di 17 $O(h) +$ aggancio $\Theta(1)$

- 3
- 5
- 6
- 7
- 10
- 12
- 13
- 15
- 16
- 17
- 18
- 20



albero
degenero



Costruzione di un ABR con n inserzioni successive

$O(n \cdot h)$

ITERATIVE - TREE SEARCH (x, k)

while ($x \neq \text{NULL} \ \&\& \ k \neq x.\text{Key}$) {

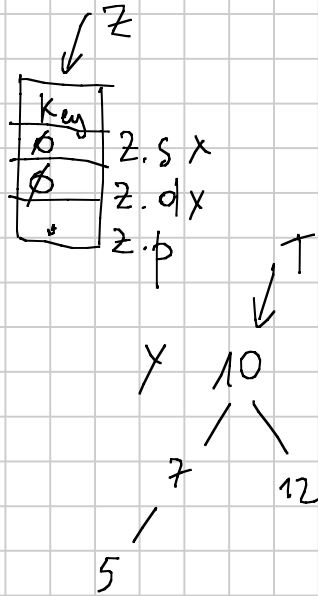
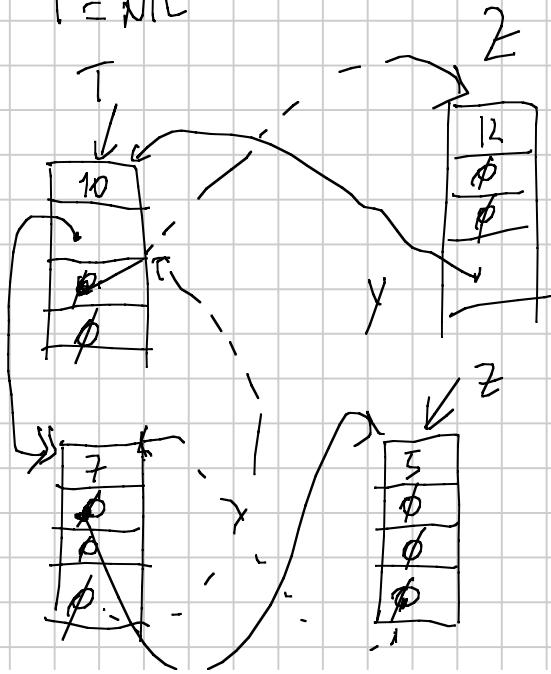
if ($k < x.\text{Key}$) $x = x.\text{left};$

 else $x = x.\text{right};$

} return $x;$

INSERZIONE (+ RICERCA)

10, 7, 5, 12
T = NIL



albero vuoto →

TREE_INSERT (T, z)

```

y = NIL;
x = T;
while (x != NULL) {
    y = x;
    if (z.Key < x.Key) x = x.left;
    else x = x.right;
}
z.p = y;
if (y = NIL) T = z;
else if (z.Key < y.Key) y.sx = z;
else y.dx = z;
    
```

Ordinamento

IN-VISITA

MIN

MAX

PRED

SUCC

ABR mantiene ordinamento

da l'insieme ordinato

della radice nei a sinistra finché possibile.
L'ultima chiave è il MIN

della radice tutto a destra

$\Theta(n)$

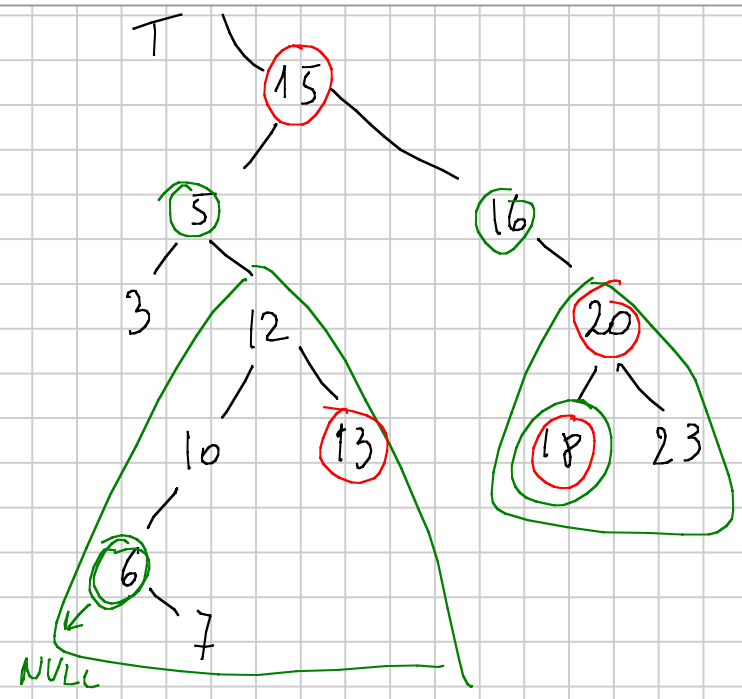
$O(h)$

$O(h)$

$SUCC(x)$
 il più piccolo elemento
 che appartiene a $T \setminus x$

$PRED(x)$
 il più grande el. $< x$

$O(h)$



$SUCC(x)$

1) Sottalbero destro di $x = NULL$

$SUCC(13) = 15$

$SUCC(18) = 20$

Nodo nel percorso radice
 foglie da cui siamo scesi
 a sinistra o più basso livello

2) Il min del sottalbero
 destro