ALBERI BINARI DI RICERCA

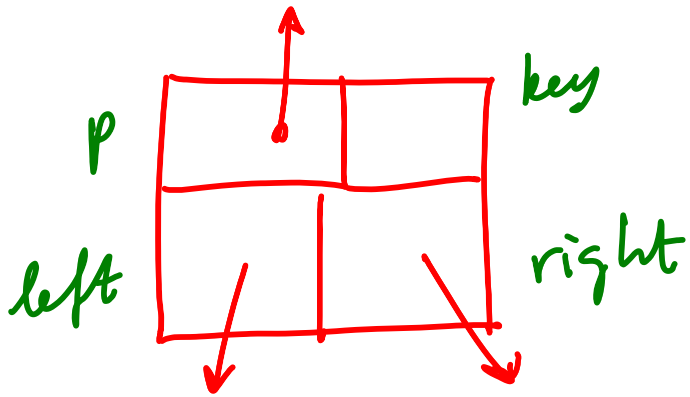
→ STRUTTURE DATI CHE SUPPORTANO LE SEGUENTI OPERAZIONI SU INSIEMI DINAMICI :

- SEARCH
- MINIMUM , MAXIMUM
- PREDECESSOR , SUCCESSOR
- INSERT
- DELETE

- SI TRATTA DI ALBERI BINARI I CUI NODI CONTENGONO GLI ATTRIBUTI

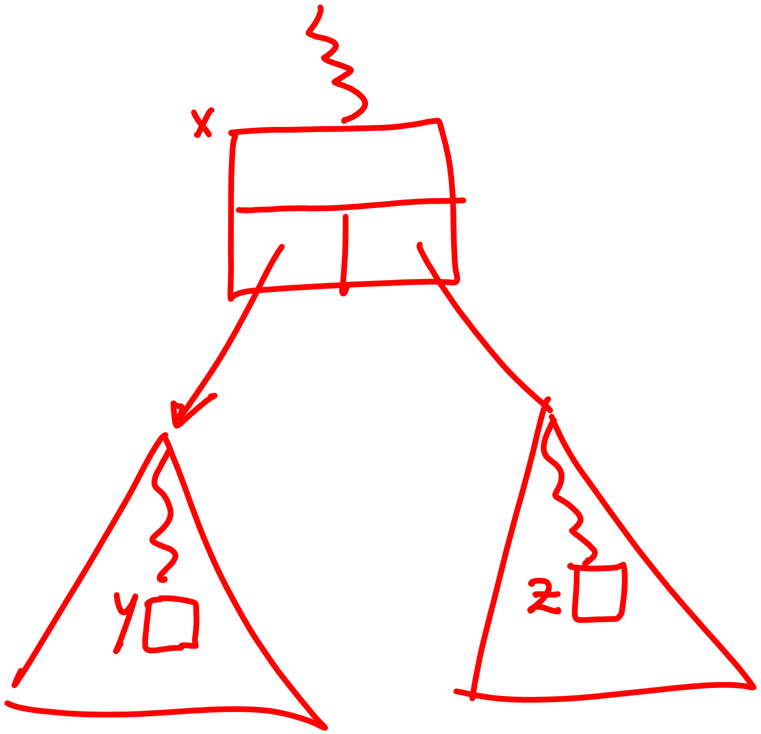
  LEFT (PUNTATORE AL FIGLIO SINISTRO / NIL)
  RIGHT (PUNTATORE AL FIGLIO DESTRO / NIL)
  P (PUNTATORE AL PADRE / NIL)
  KEY (CHIAVE)

  + EVENTUALI CAMPI SATELLITI



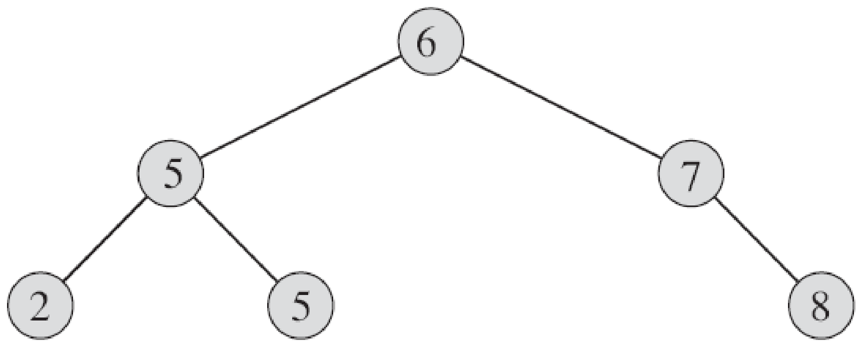
- IL NODO RADICE (ROOT) E' L'UNICO NODO IL CUI ATTRIBUTO PADRE E' NIL

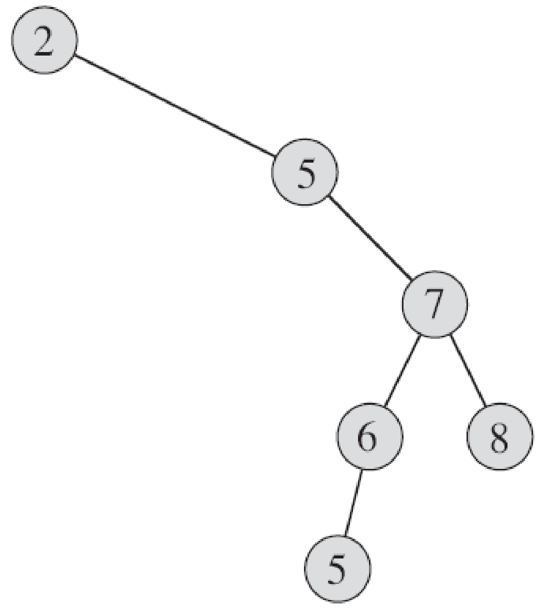
- SODDISFANO LA PROPRIETA' DEGLI ALBERI DI RICERCA:



$$y.key \leq x.key \leq z.key$$

ESEMPI





- GRAZIE ALLA PROPRIETÀ DEGLI ALBERI BINARI DI RICERCA LE CHIAVI DI UN ALBERO BINARIO DI RICERCA POSSONO ESSERE FACILMENTE ELENCATE IN MANIERA ORDINATA MEDIANTE UN ATTRAVERSAMENTO SIMMETRICO (INORDER)

```
INORDER-TREE-WALK(x)
if x ≠ NIL
    INORDER-TREE-WALK(x.left)
    print x.key
    INORDER-TREE-WALK(x.right)
```

TEOREMA SE x È LA RADICE DI UN SOTTOALBERO BINARIO DI n NODI, LA CHIAMATA INORDER-TREE-WALK(x) HA COMPLESSITÀ $T(n) = \Theta(n)$.

DIM. – OVVIAMENTE $T(n) = \Omega(n)$, IN QUANTO CIASCUNO DEGLI n NODI VIENE VISITATO.

– VERIFICHIAMO CHE VALE ANCHE $T(n) = O(n)$

CASO BASE

– $T(0) = c$ (TEMPO RICHIESTO DAL TEST $x \neq \text{NIL}$)

PASSO RICORSIVO ($x.left$ HA k NODI)

– $T(n) \leq T(k) + T(n-k-1) + d$ (d COSTANTE)

DIMOSTRIAMO CHE

$$\begin{cases} T(0) = c \\ T(n) \leq T(k) + T(n-k-1) + d, & n \geq 1, \text{ QUALCHE } k \end{cases}$$

HA SOLUZIONE $T(n) = O(n)$ FACENDO VEDERE

CHE VALE $T(n) \leq (c+d)n + c$, PER OGNI $n \geq 0$.

- $n = 0 \rightarrow T(n) = c$, $(c+d)n + c = c \implies$ O.K.
- $n \geq 1 \rightarrow T(n) \leq T(k) + T(n-k-1) + d$ (PER QUALCHE k)

$$\leq \big((c+d)k + c\big) + \big((c+d)(n-k-1) + c\big) + d$$

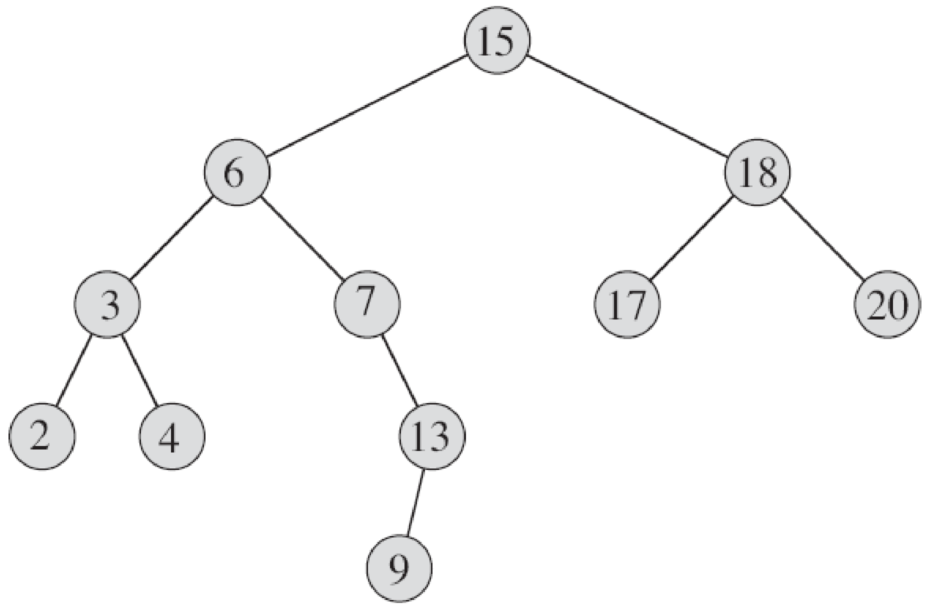
$$= (c+d)n + c \,.$$

$\blacksquare$

RICERCA DI UNA CHIAVE

VERSIONE RICORSIVA

```
TREE-SEARCH(x, k)
if x == NIL or k == x.key
    return x
if k < x.key
    return TREE-SEARCH(x.left, k)
else return TREE-SEARCH(x.right, k)
```



VERSIONE ITERATIVA

```
ITERATIVE-TREE-SEARCH(x, k)
while x ≠ NIL and k ≠ x.key
    if k < x.key
        x = x.left
    else x = x.right
return x
```

COMPLESSITA': $O(h)$ (h ALTEZZA)

MINIMO

VERSIONE RICORSIVA

TREE-MINIMUM-RECURSIVE(x)

if x.left == NIL **then**

  return x

else TREE-MINIMUM-RECURSIVE(x.left)

VERSIONE ITERATIVA

TREE-MINIMUM(x)

while $x.left \neq$ NIL

  $x = x.left$

return x

COMPLESSITÀ: $O(h)$ (h ALTEZZA)

MASSIMO

VERSIONE RICORSIVA

TREE-MAXIMUM-RECURSIVE(x)

    if x.right == NIL **then**

        return x

    else TREE-MAXIMUM-RECURSIVE(x.right)

VERSIONE ITERATIVA

TREE-MAXIMUM(x)

1 **while** $x.right \neq$ NIL

2     $x = x.right$

3 **return** x

COMPLESSITA': $O(h)$ (h ALTEZZA)

SUCCESSORE E PREDECESSORE

```
TREE-SUCCESSOR(x)
if x.right ≠ NIL
    return TREE-MINIMUM(x.right)
y = x.p
while y ≠ NIL and x == y.right
    x = y
    y = y.p
return y
```

```
TREE-PREDECESSOR(x)
if x.left ≠ NIL
    return TREE-MAXIMUM(x.left)
y = x.p
while y ≠ NIL and x == y.left
    x = y
    y = y.p
return y
```

COMPLESSITA': $O(h)$ (h ALTEZZA)

ESERCIZI

12.1-1

For the set of $\{1, 4, 5, 10, 16, 17, 21\}$ of keys, draw binary search trees of heights 2, 3, 4, 5, and 6.

12.1-5

Argue that since sorting n elements takes $\Omega(n \lg n)$ time in the worst case in the comparison model, any comparison-based algorithm for constructing a binary search tree from an arbitrary list of n elements takes $\Omega(n \lg n)$ time in the worst case.

12.2-1

Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could *not* be the sequence of nodes examined?

a. 2, 252, 401, 398, 330, 344, 397, 363.

b. 924, 220, 911, 244, 898, 258, 362, 363.

c. 925, 202, 911, 240, 912, 245, 363.

d. 2, 399, 387, 219, 266, 382, 381, 278, 363.

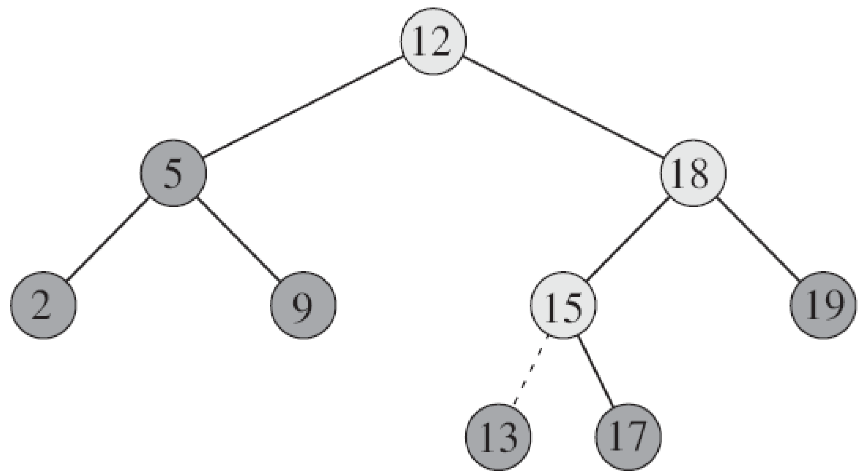
e. 935, 278, 347, 621, 299, 392, 358, 363.

12.2-4

Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest possible counterexample to the professor's claim.

INSERIMENTO

```
TREE-INSERT(T, z)
y = NIL
x = T.root
while x ≠ NIL
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == NIL
    T.root = z        // tree T was empty
elseif z.key < y.key
    y.left = z
else y.right = z
```

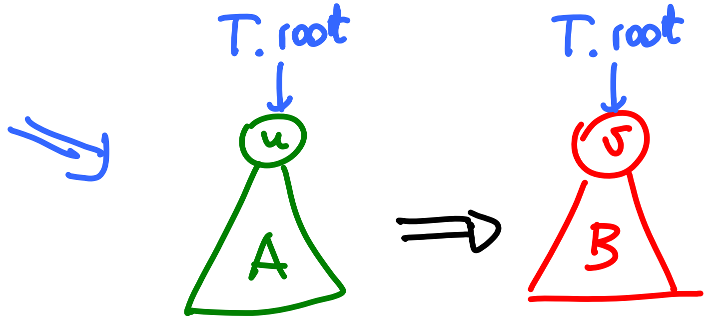


COMPLESSITA': $O(h)$ (h ALTEZZA)

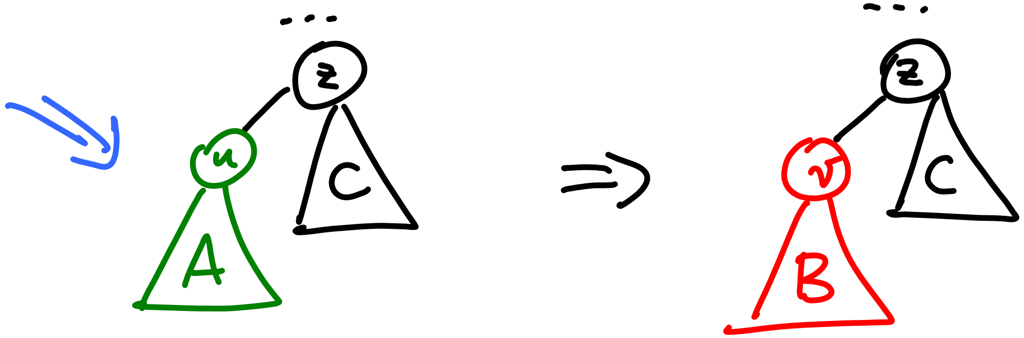
CANCELLAZIONE

TRANSPLANT(T, u, v) (SOSTITUISCE v AL POSTO DI u)

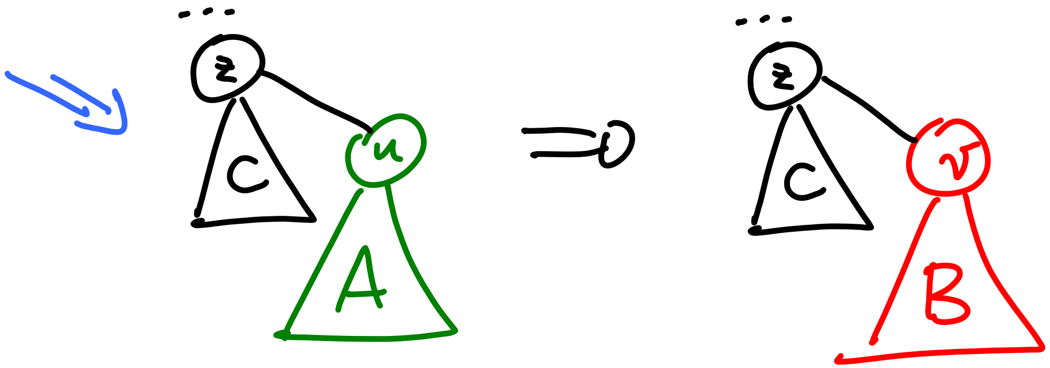
if $u.p == \text{NIL}$
  $T.root = v$



elseif $u == u.p.left$
  $u.p.left = v$



else $u.p.right = v$

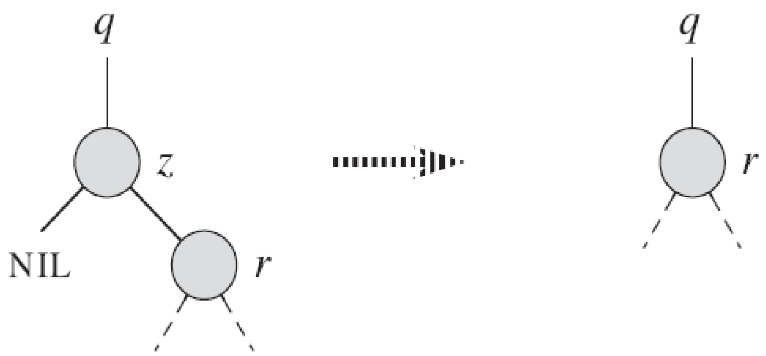


if $v \neq \text{NIL}$
  $v.p = u.p$

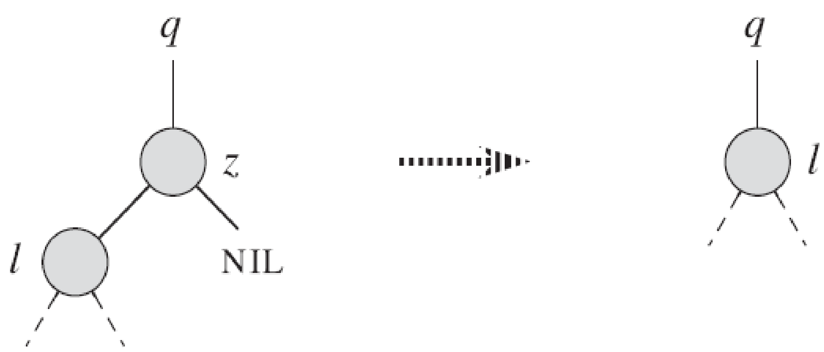
COMPLESSITÀ: $\Theta(1)$

TREE-DELETE(T, z)

```
if z.left == NIL
    TRANSPLANT(T, z, z.right)
```

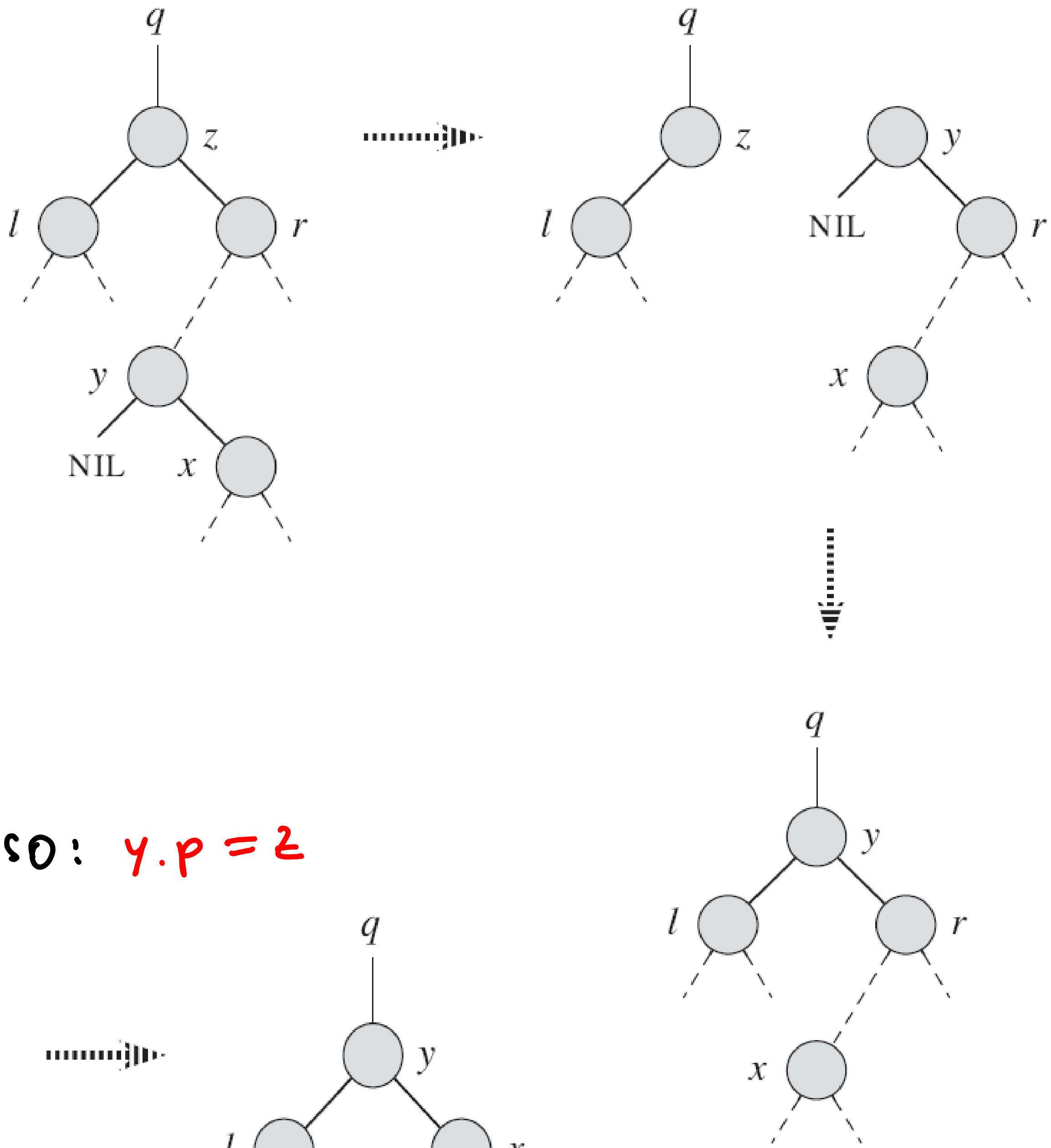



```
elseif z.right == NIL
    TRANSPLANT(T, z, z.left)
```

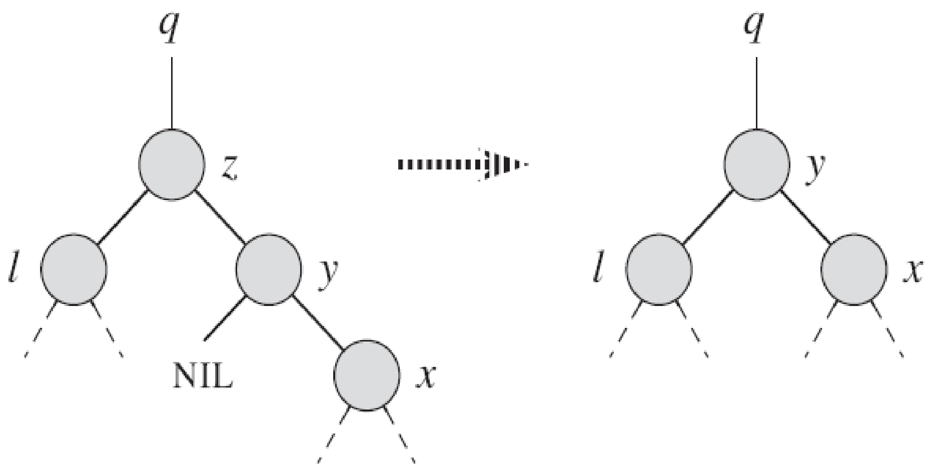


```
else y = TREE-MINIMUM(z.right)
    if y.p ≠ z
        TRANSPLANT(T, y, y.right)
        y.right = z.right
        y.right.p = y
    TRANSPLANT(T, z, y)
    y.left = z.left
    y.left.p = y
```

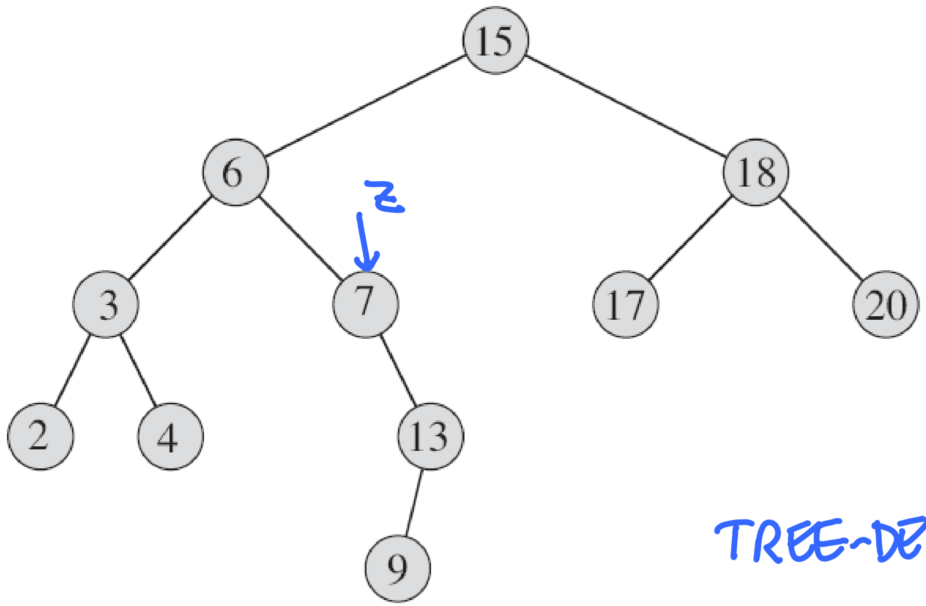
CASO: $y.p \neq z$



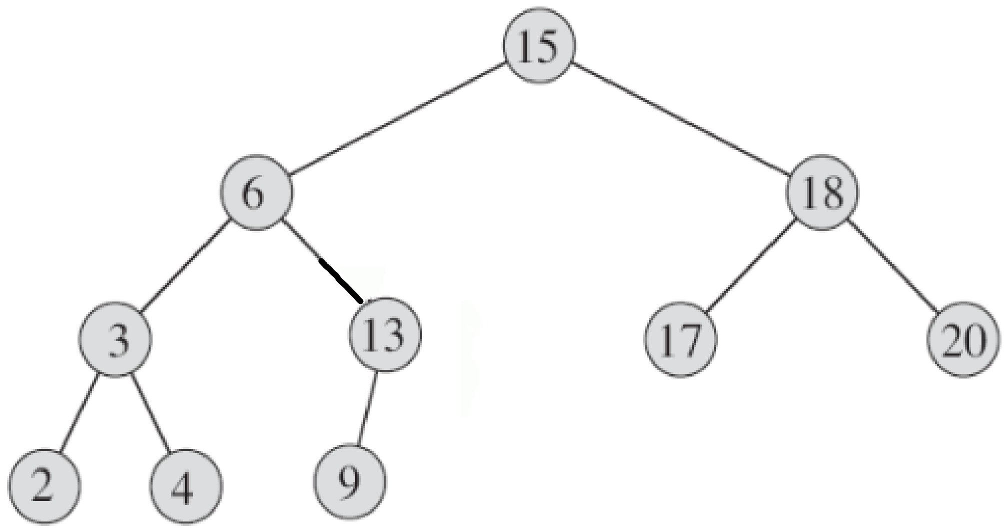
CASO: $y.p = z$

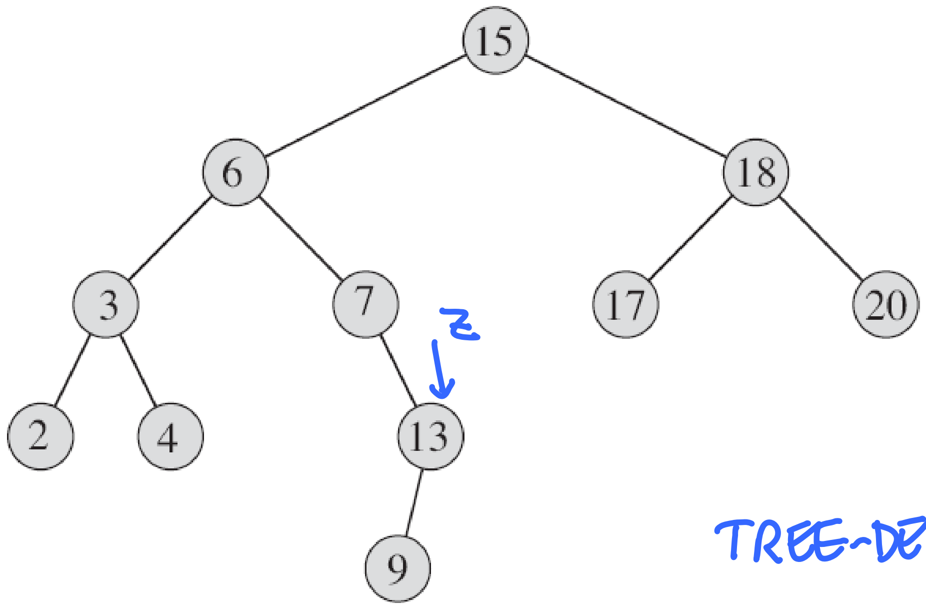


ESEMPI:

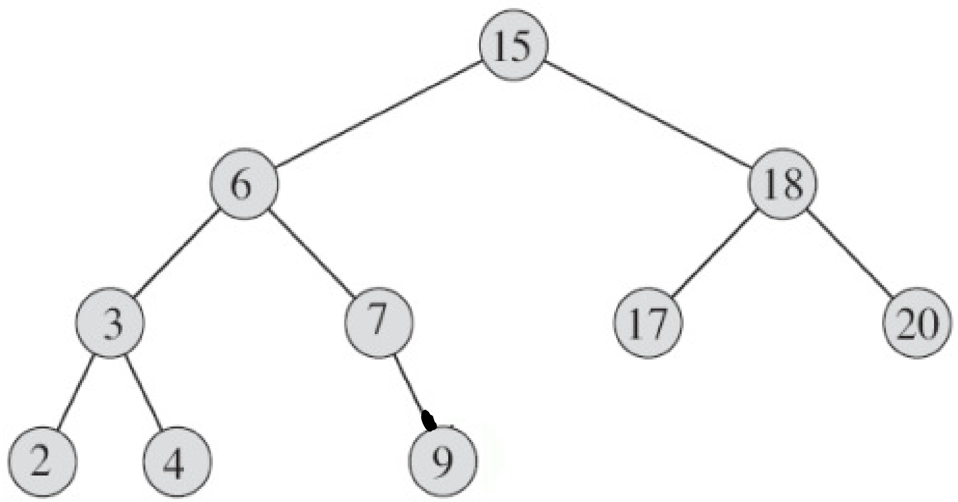


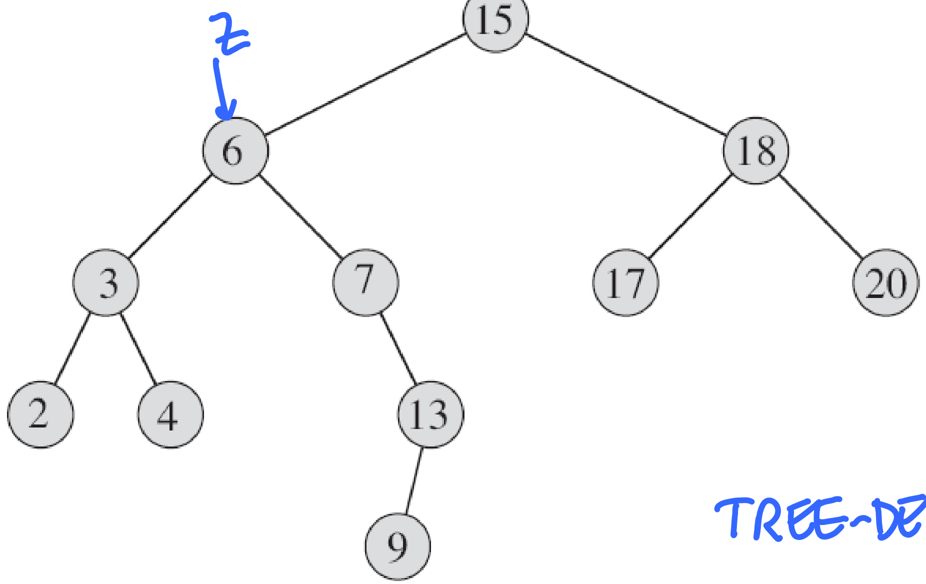
TREE-DELETE(T, z)



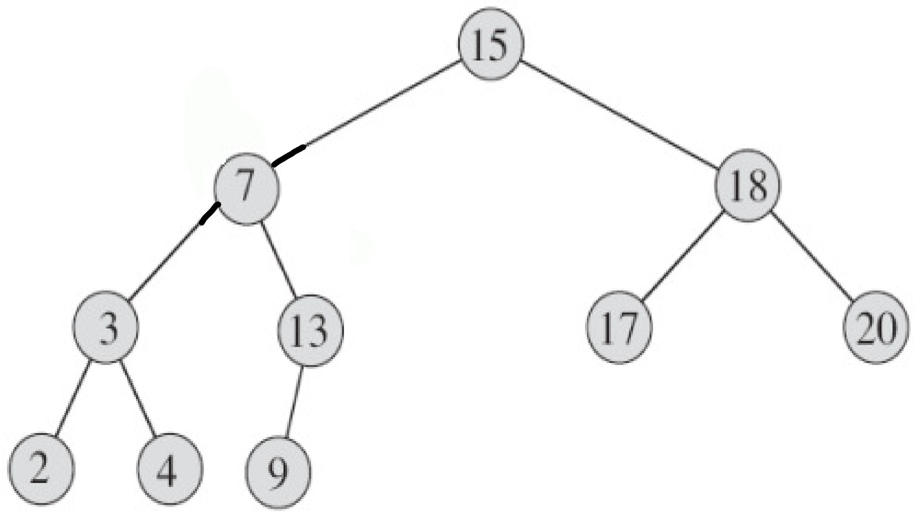


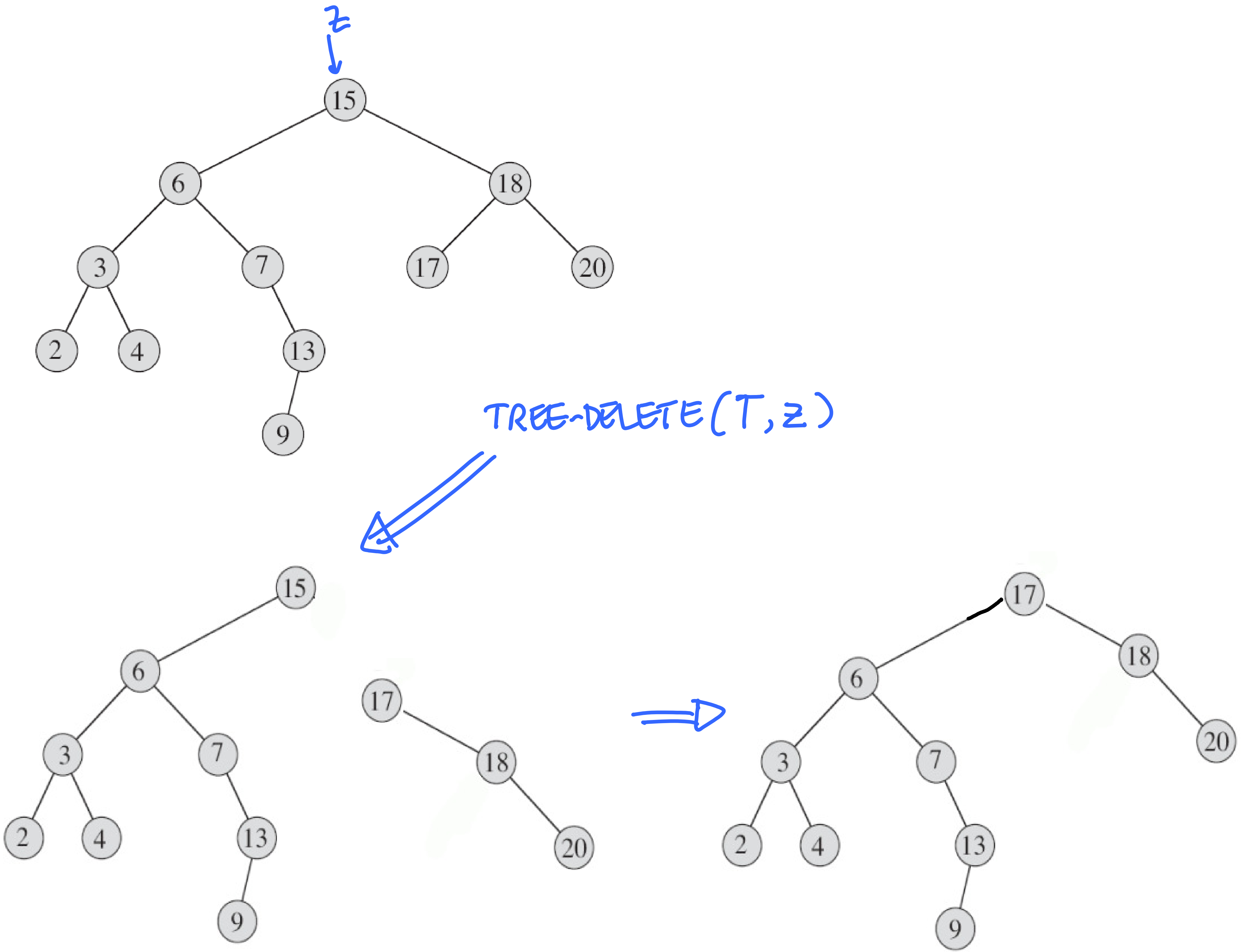
TREE-DELETE(T, z)





TREE-DELETE(T, z)




z
TREE-DELETE(T, z)

COMPLESSITÀ CANCELLAZIONE: $O(h)$ (h ALTEZZA)

- UN ALBERO BINARIO COMPLETO CON n NODI HA ALTEZZA $\Theta(\lg n)$
- MA SE È UNA CATENA LINEARE HA ALTEZZA $\Theta(n)$
- UN ALBERO BINARIO RANDOM HA ALTEZZA ATTESA $O(\lg n)$
- VEDREMO UNA VARIANTE DEGLI ALBERI BINARI DI RICERCA CHE ASSICURANO UN'ALTEZZA $\Theta(\lg n)$ NEL CASO PEGGIORE (ALBERI ROSSO-NERI)