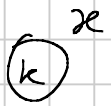
Alberi binari: Implementazione di dizionari ordinati di Ricerca

- struttura dati concatenata
- ogni nodo x rappresenta un elemento del dizionario

  $x.key$
  $x.dati$
  $x.p$
  $x.left$
  $x.right$

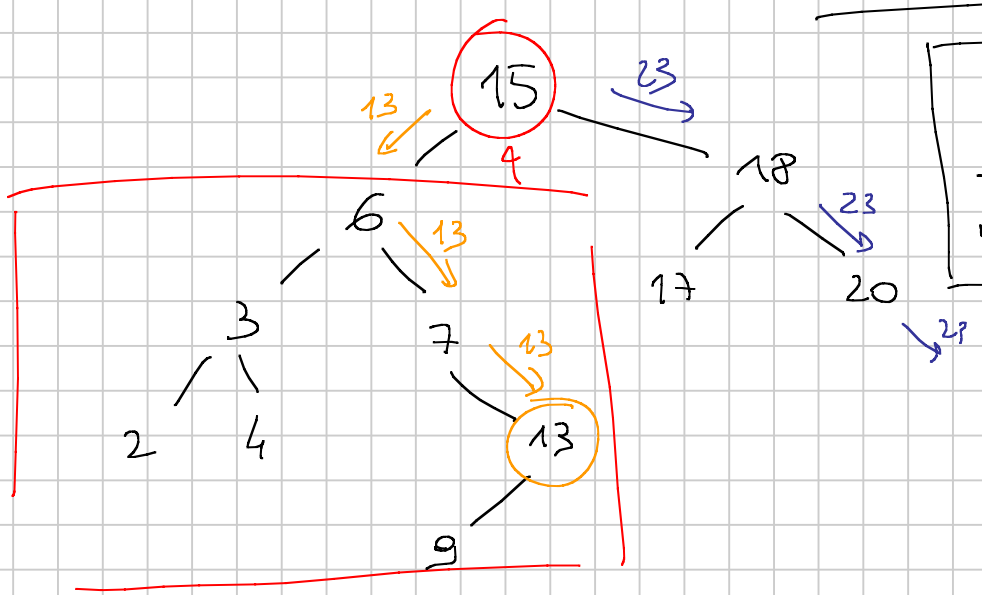
- chiavi distribuite in modo da verificare la PROPRIETÀ di ABR

$\forall x$ in un ABR



$\forall$ nodo y nel sottoalbero sinistro di x, $y.key < x.key$

$\forall$ nodo y nel sottoalbero destro di x, $y.key > x.key$



La visita simmetrica di un ABR fornisce la sequenza ordinata degli elementi del dizionario

INTERROGAZIONI

```
TREE-SEARCH (x, k)
if (x == NIL || x.key == k) return x
if (x.key > k) return TREE-SEARCH (x.left, k);
else return TREE-SEARCH (x.right, k);
```

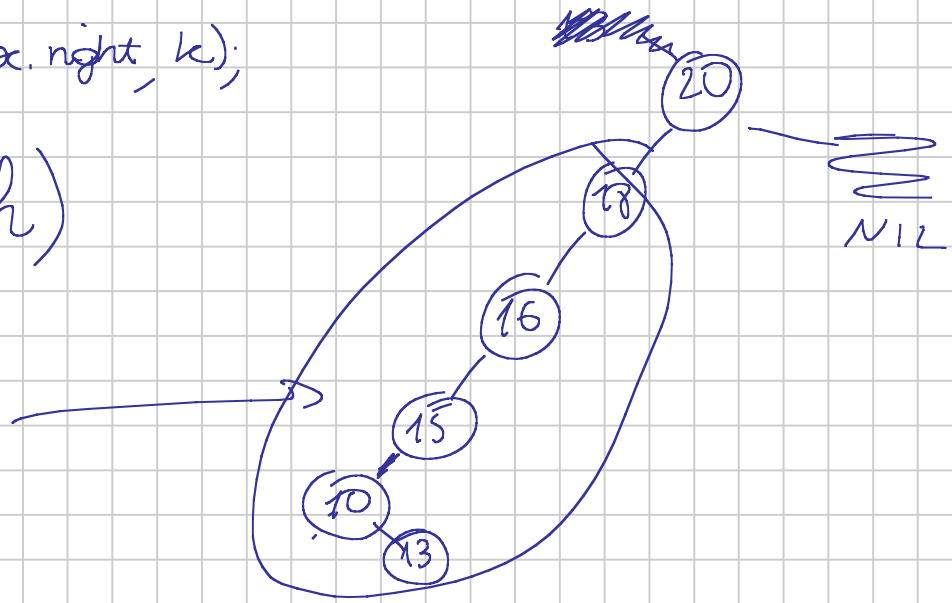
$$T(n) = O(h)$$

$$h = O(n)$$

CASO PESSIMO $T(n) = \Theta(n)$

CASO OTTIMO $T(n) = \Theta(1)$

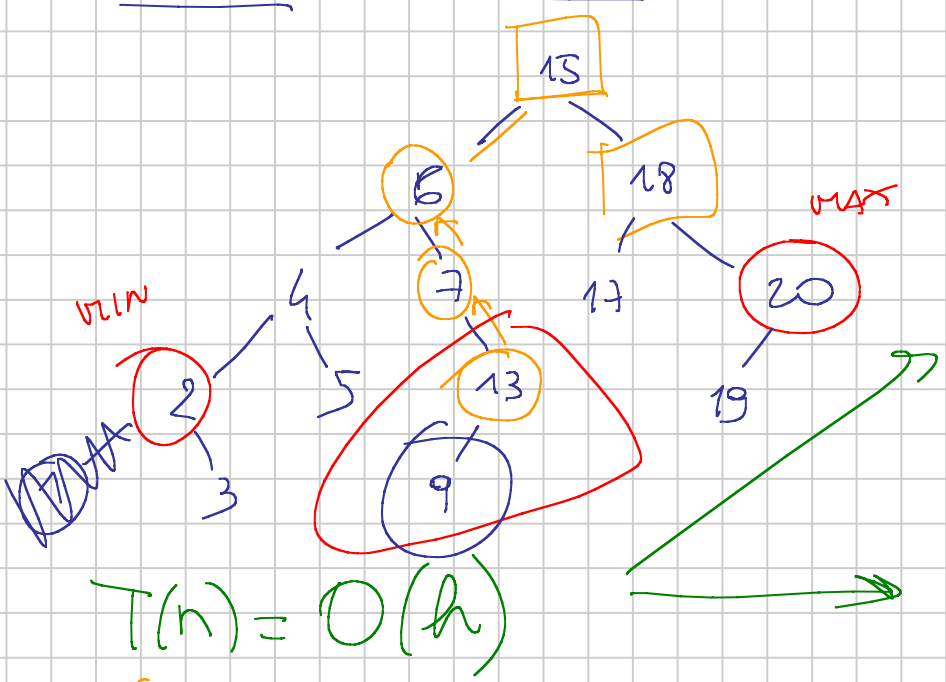
CASO MEDIO $T(n) = O(\log n)$



ITERATIVE-TREE-SEARCH (x, k)

```
while (x ≠ NIL && x.key ≠ k) {
        if (x.key > k)  x = x.left;
        else  x = x.right;
}
return x;
```

MINIMO e MASSIMO



$T(n) = O(h)$

SUCC(18) = 19
SUCC(6) = 7
SUCC(15) = 17

SUCC(13) = 15
SUCC(5) = 6
SUCC(9) = 13

```
TREE-MINIMUM (x)
if (x == NIL) return NIL;
while (x.left ≠ NIL) x = x.left;
return x;
```

```
TREE-MAXIMUM (x)
if (x == NIL) return NIL;
while (x.right ≠ NIL) x = x.right;
return x;
```

SUCCESSORE

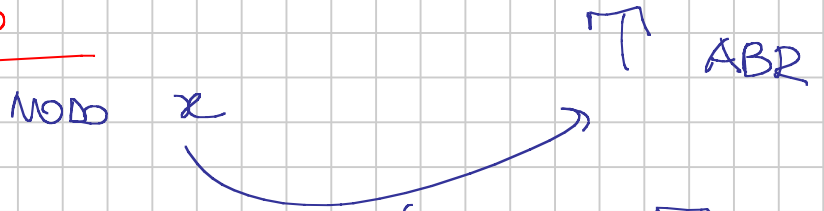
```
TREE-SUCCESSOR(x)                    // nodo y con la più piccola chiave che
                                     // sia > x.key.
if(x==NIL) return NIL;
if (x.right ≠ NIL) return TREE-MINIMUM (x.right);
// il successore di x è l'antenato più vicino a x, che contiene x nel suo
   sottoalbero sinistro.
y = x.p
while( y ≠ NIL && y.right == x ) {
        x = y
        y = y.padre;
}
return y
```

$T(n) = O(h)$

RESTITUISCE NIL se x è il nodo di chiave MAX

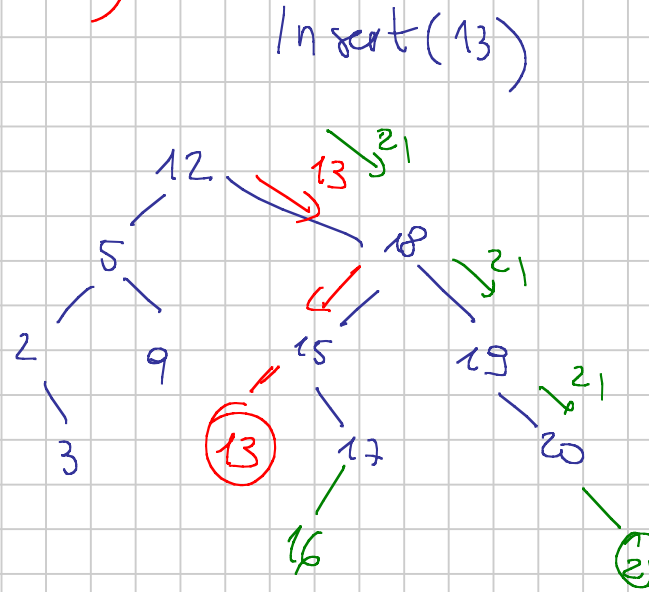
OPERAZIONI di MODIFICA (inserimenti e cancellazioni)

INSERIMENTO



x sarà una nuova foglia di T

x.key
x.dati
x.left = NIL
x.right = NIL



TREE-INSERT (T, z)

z.left == z.right == NIL

x = T.root
y = NIL // puntatore inseguitore (padre di x)
while (x ≠ NIL) {
    y = x
    if (z.key < x.key) x = x.left;
    else x = x.right
}
// x è NIL, y è il padre di x, y dovrà diventare il padre del nuovo nodo z
z.p = y
if (y == NIL) T.root = z // T era vuoto
else if (z.key < y.key) y.left = z;
    else y.right = z;

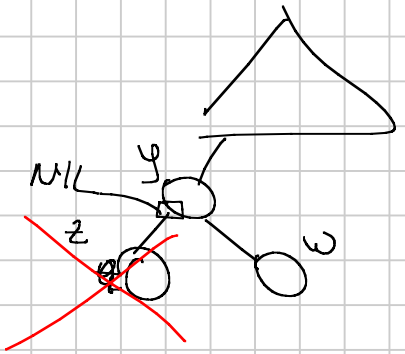
O(h)

O(1)

T(n) = O(h) = O(n)

CANCELLAZIONE

TREE-DELETE (z)



3 CASI

1 **z è una foglia**

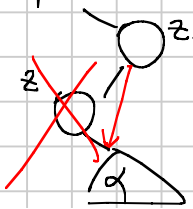
si sostituisce il riferimento a z nel padre di z, con un riferimento a NIL
poi si elimina z

$T(n) = \Theta(1)$

2 **z ha un solo figlio**

si sostituisce il riferimento a z nel nodo padre di z (z.p)
con un riferimento all'unico figlio di z,
poi si elimina z

$T(n) = \Theta(1)$



③ z ha due figli

$O(h)$

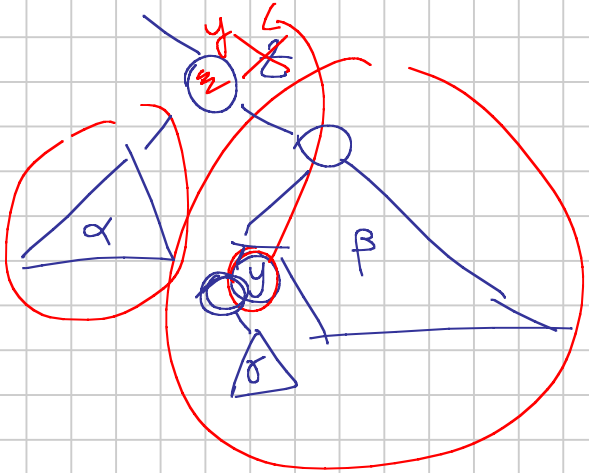
→ si cerca il successore y di z (MINIMO del sottoalbero dx)

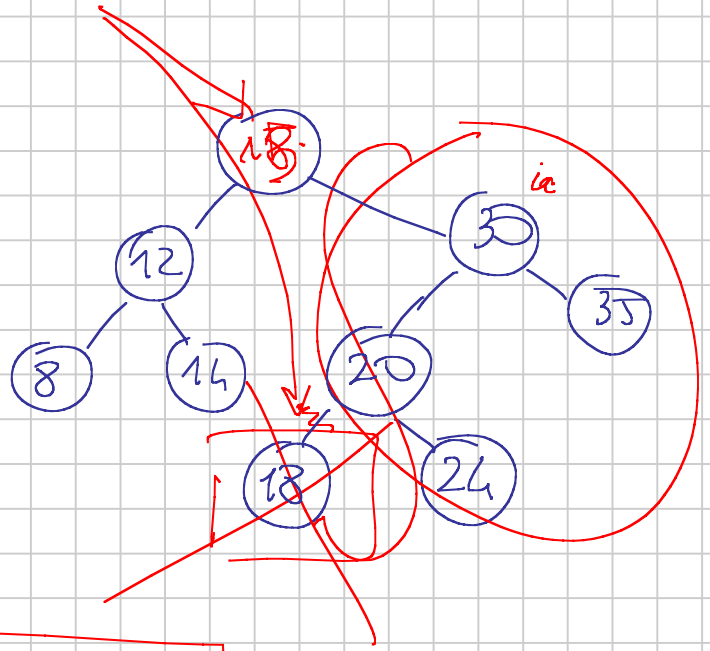
$O(1)$

→ si sostituiscono chiave e dati di z, con chiave e dati di y

→ si elimina y (① o ②)

[y NON ha figlio sx]

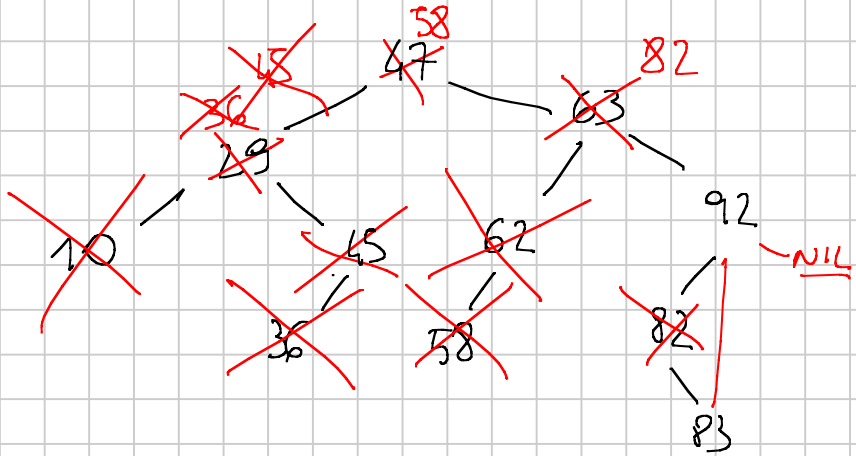




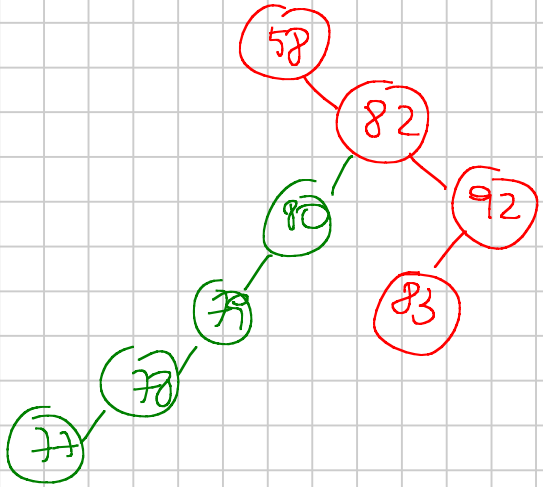
$$T(n) = O(h) = O(n)$$

ESEMPIO

$S = \{47, 29, 45, 63, 62, 92, 10, 36, 58, 82, 83\}$



DELETE: 29, 47, 63, 36, 62, 10, 45



INSERT(80, 79, 78, 77)