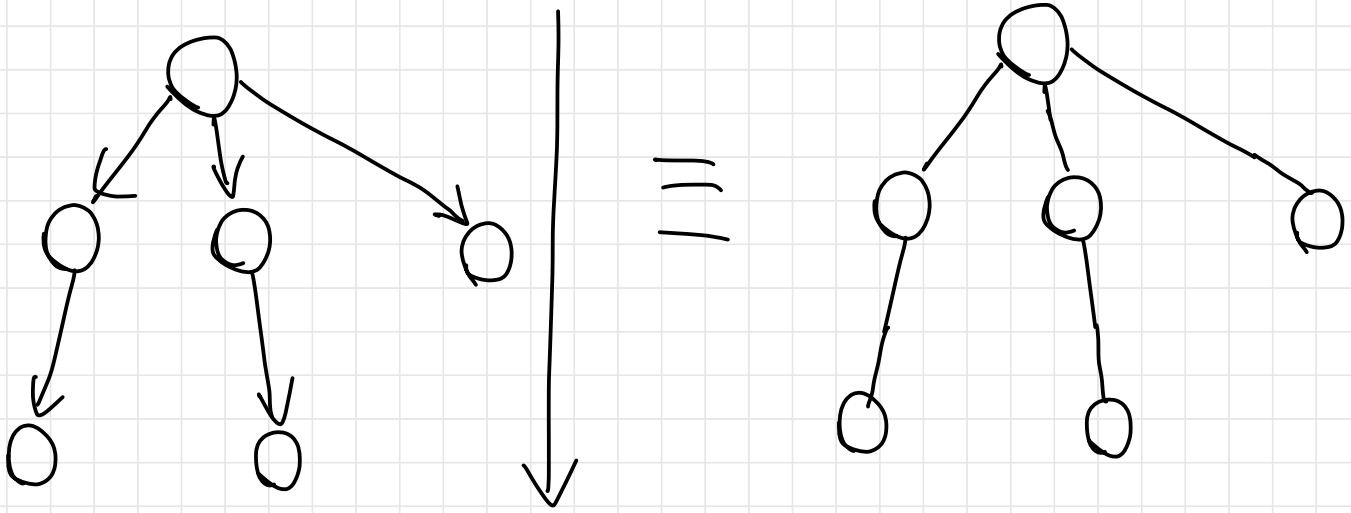
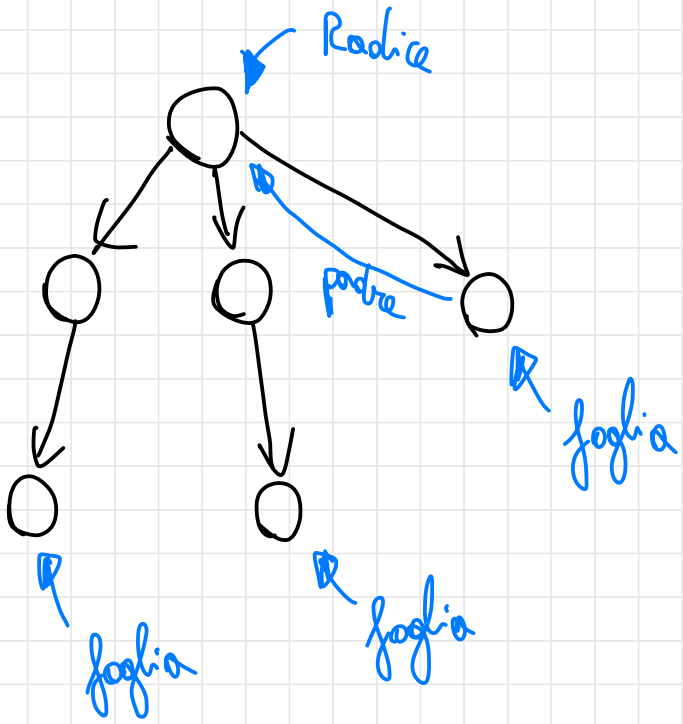
Alberi

Un albero è un particolare grafo orientato

Esempio





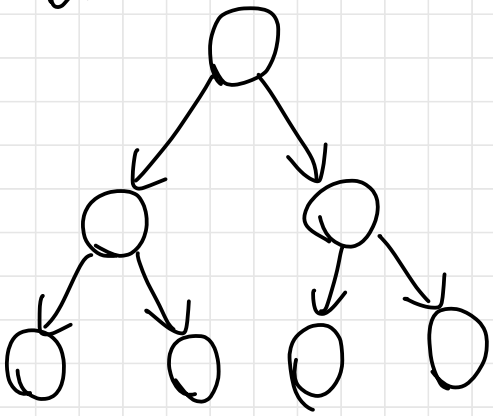
Nomenclatura:

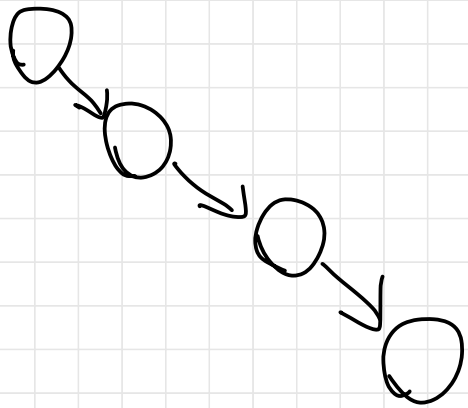
- Radice
- Foglia
- Padre
- Figlio

N.B.: La radice è l'unico nodo senza padre

Albero binario: albero dove ogni nodo ha <u>al più</u> 2 figli

Esempi BT:





Struttura dati di un nodo:

A.key (assumiamo sia numerico)

A.left → puntatori al figlio sx / dx
A.right

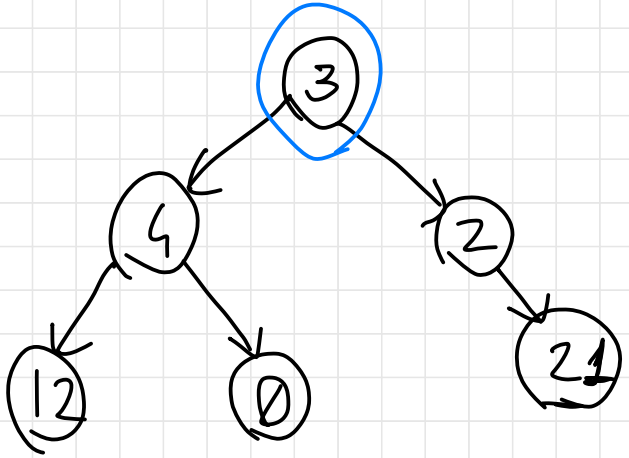
A.p → puntatore al padre

A.root → puntatore alla radice

Operazione di visita : InOrder , PreOrder , PostOrder

↓

Visitare ogni nodo dell'albero




```
InOrder(T)
    InOrder(T.left)  ✓
    print(T.key)     ✓
    InOrder(T.right) ✓
```

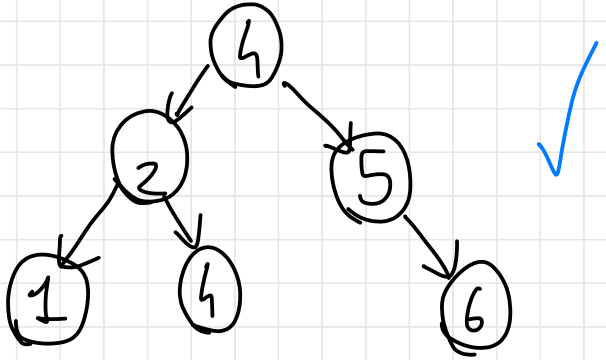
Cambio in PreOrder e PostOrder

Stampe : 12, 4, 0, 3, 2, 21

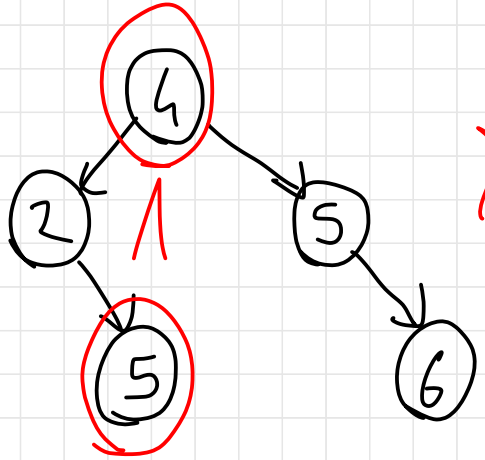
Albero binario di ricerca: è un albero binario dove preso un nodo n: $\forall$ nodo nel sottoalb. sx di n le chiavi $\leq n.key$

$\forall$ nodo nel sottoalbero dx, le chiavi $\geq n.key$

Esempi BST



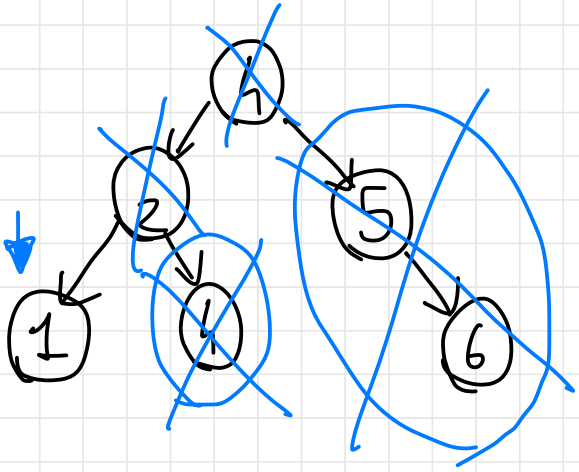
✓



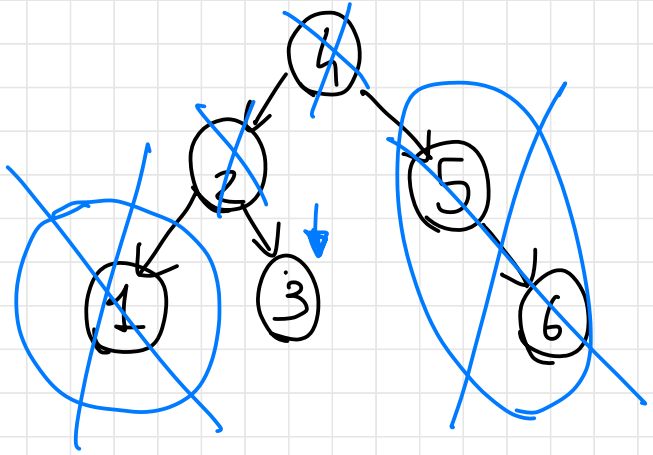
✗

Ricerca:

search(T, 1) = nodo(1)

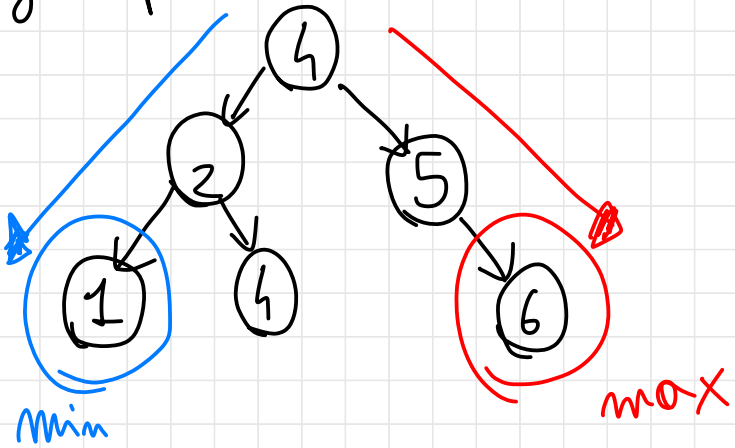


search(T, 3) 3 > 2



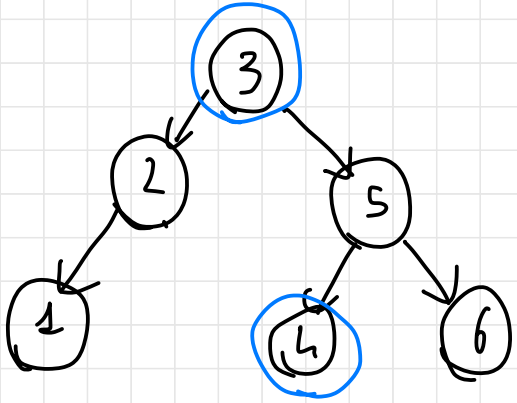
min: foglia più a sx

max: foglia più a dx



Successore:

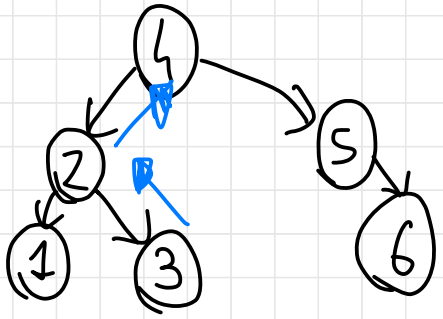
succ(3) ?



Vado nel sottoalb. dx e trovo il min.

Successore :

succ(3) ? 4



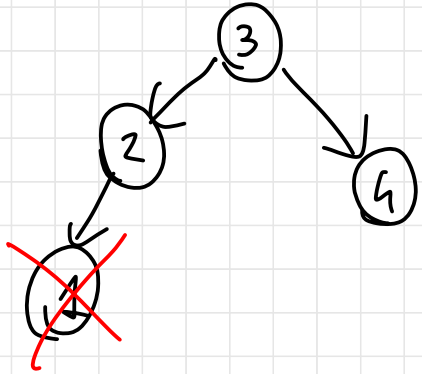
Quando non c'è figlio dx, risalgo fino a che il nodo corrente non è il figlio sx ritorno il padre

Inserimento : ricerca del nodo e inserimento dove dovrebbe essere il nodo

Cancellazione:

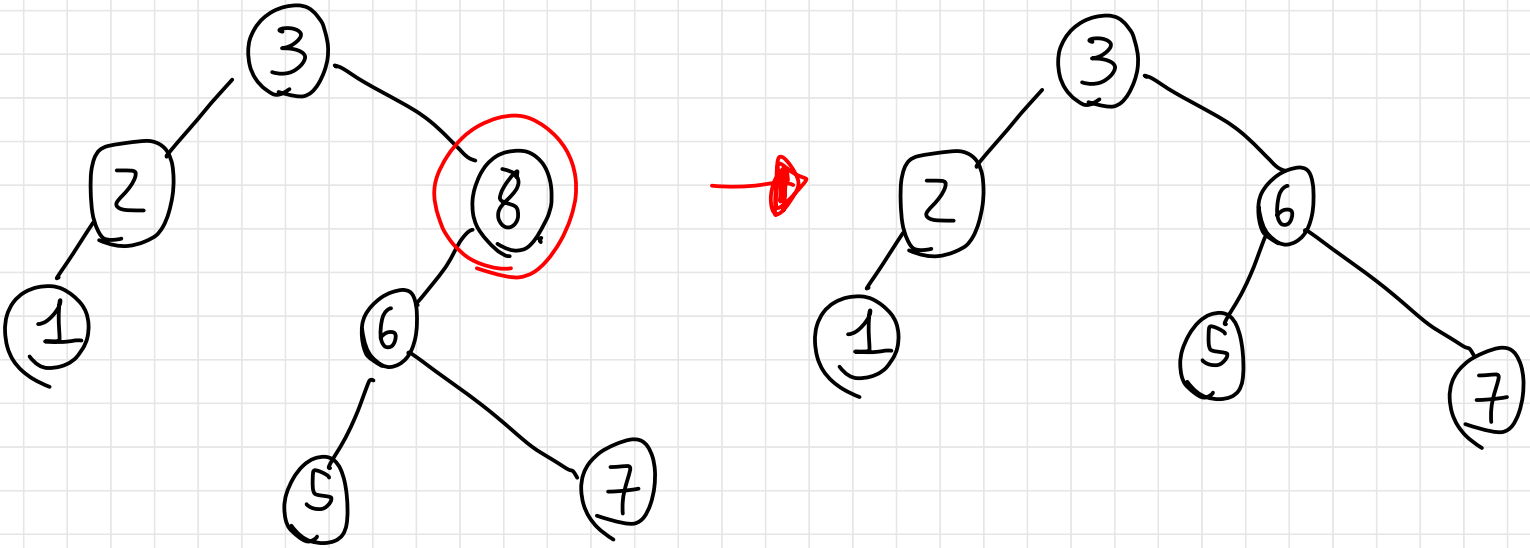
Caso 1: l'elem. da cancellare non ha figli

→ elimino l'elem.

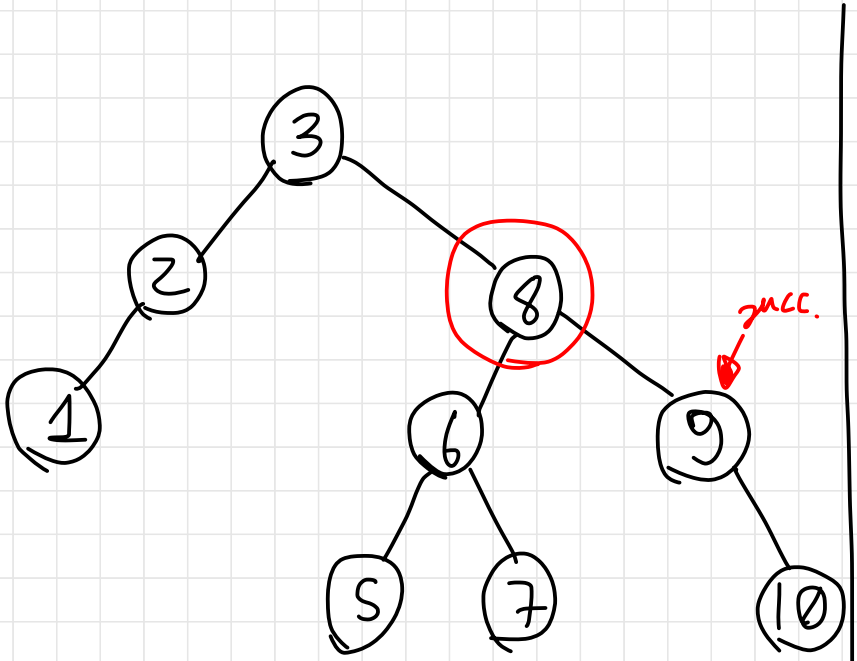


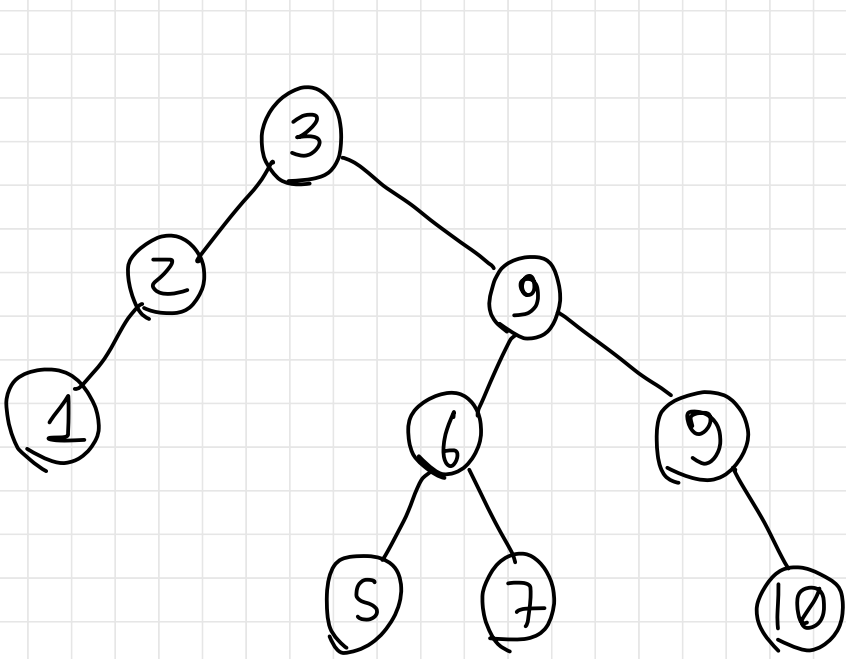
Caso 2: L'elem. ha un figlio

→ l'elem. verrà sostituito dal figlio

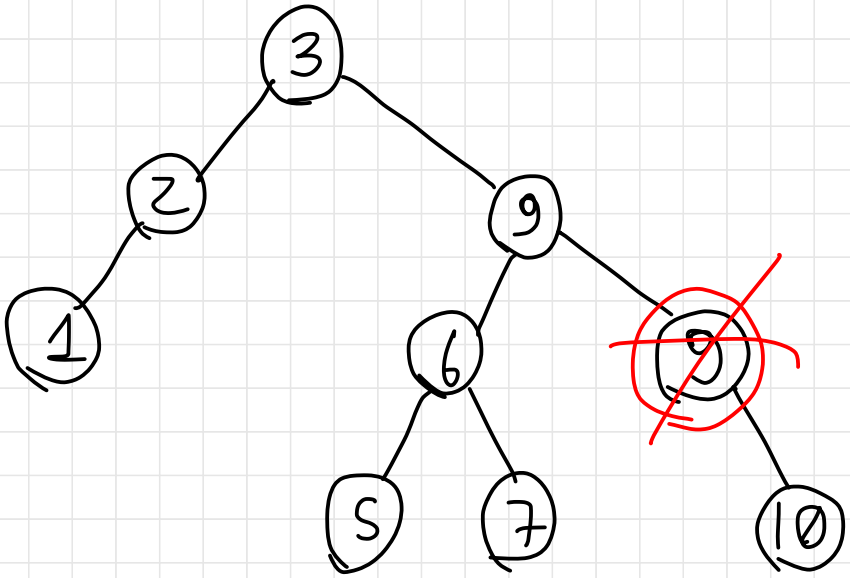


Caso 3: l'elem ha due figli





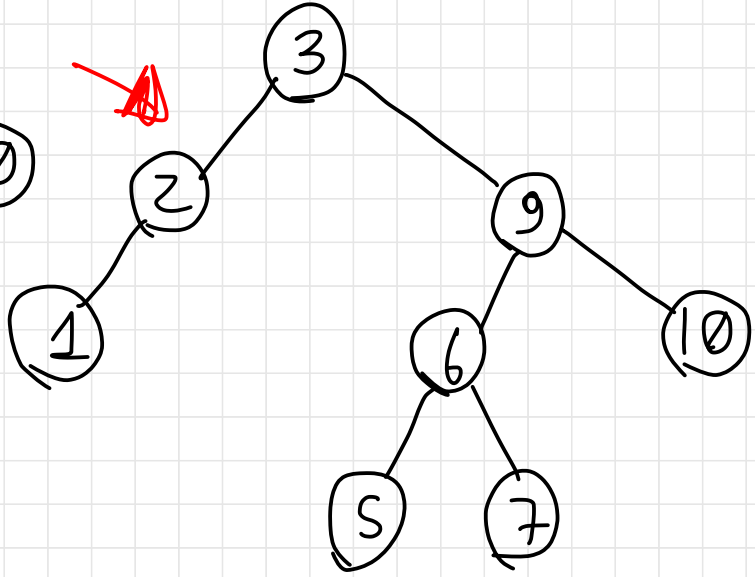
successore s di un elem x
non può avere figlio sx f



$a.key < f.key < x.key$

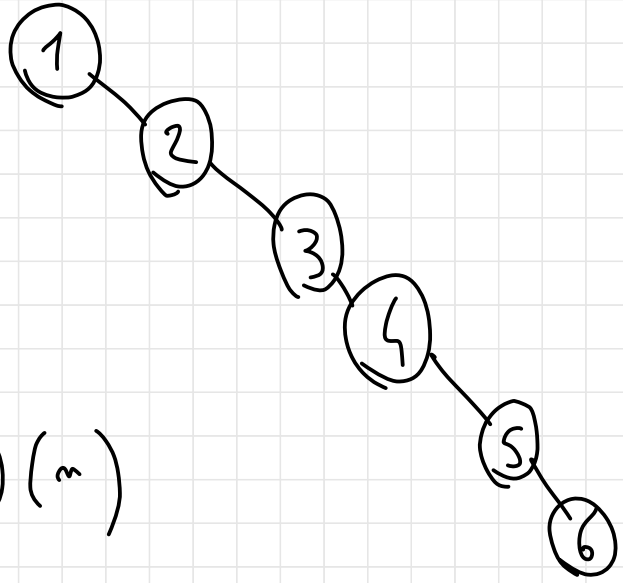
impossibile per def. di succ.

successore s di un elem x non può avere figlio sx f



Tutti gli algoritmi su BST : $O(h)$, dove h è l'altezza dell'albero

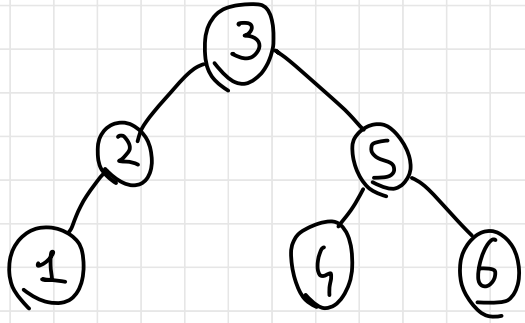
BST sbilanciato :



$h = n$

$\Rightarrow O(n)$

BST bilanciato :



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

es.:

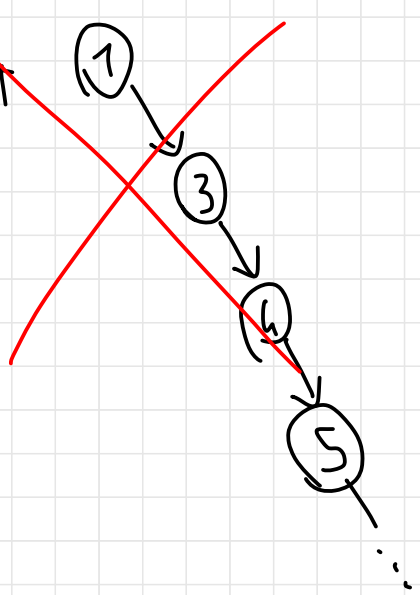
Dato un array A, costruire un BST bilanciato

$A = [5, 3, 1, 6, 7, 10, 4]$

↓ sort

$A = [1, 3, 4, 5, 6, 7, 10]$

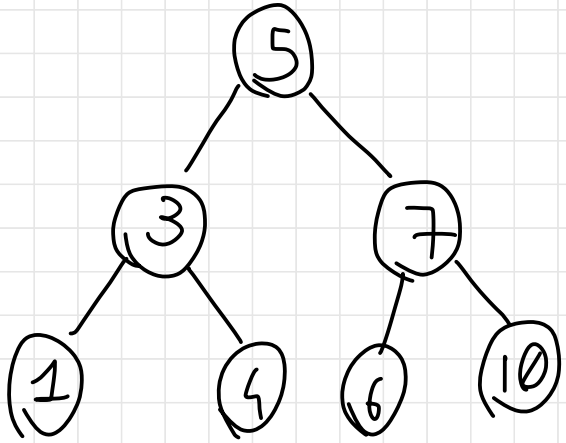
inserisco in ordine in un BST vuoto



$A = [5, 3, 1, 6, 7, 10, 4]$

↓ sort

$A = [1, 3, 4, 5, 6, 7, 10]$



caso pessimo

↓

Algoritmo:

ordina(A) $(\Theta(n \log(n)))$

prendo l'elem. centrale di A

inserisco in BST

ripeto per i sottoarr. dx e sx

Ad ogni inserimento i-esimo, inseriamo un elem. in un BST grande i : $\log(i)$

$$T(n) = \Theta(n \log(n)) + \sum_{i=1}^{n} \log(i) = \Theta(n \log(n)) + \log(n!)$$

$$= \Theta(n \log(n)) + \Theta(n \log(n)) = \Theta(n \log(n))$$

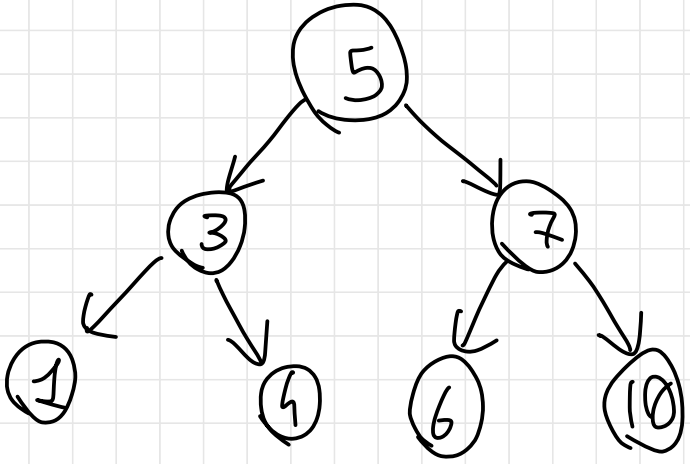
↑ approx. di Stirling

$A = [5, 3, 1, 6, 7, 10, 4]$

ordino A
build(A)

↓ sort

$A = [1, 3, 4, 5, 6, 7, 10]$



build:

prendo l'elem. centrale

creo un nodo con key
= elem. centrale

n.left ← build(sottoarr. sx)

n.right ← build(sottoarr dx)

$$T_{build}(n) = 2\,T\left(\frac{n}{2}\right) + \Theta(1)$$

$a = 2$, $b = 2$

MT: $n^{\log_b a} = n$

$\Rightarrow \Theta(n)$

build:

prendo l'elem. centrale

creo un nodo con key = elem. centrale

n.left ← build(sottoarr. sx)

n.right ← build(sottoarr dx)

Complessità:

$$T(n) = \underbrace{\Theta(n \log(n))}_{\text{sort}} + \underbrace{\Theta(n)}_{\text{build}} = \Theta(n \log(n))$$

es.: Minimo Antenato Comune (MCA)

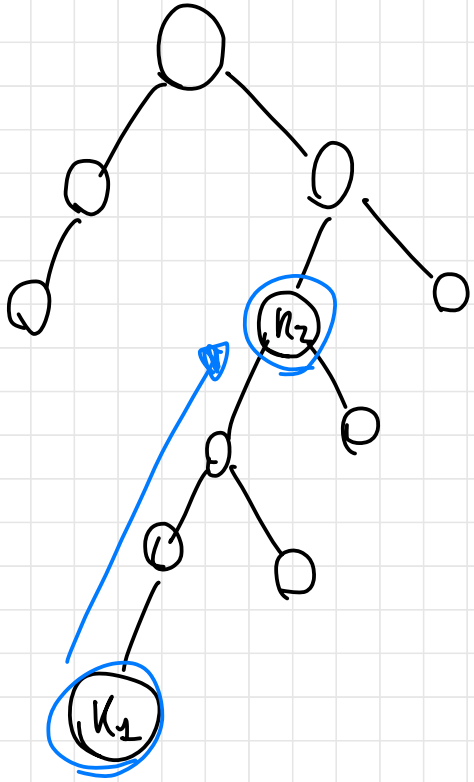
Input: due chiavi k_1, k_2, BST — Output: MCA dei nodi con chiavi k_1 e k_2

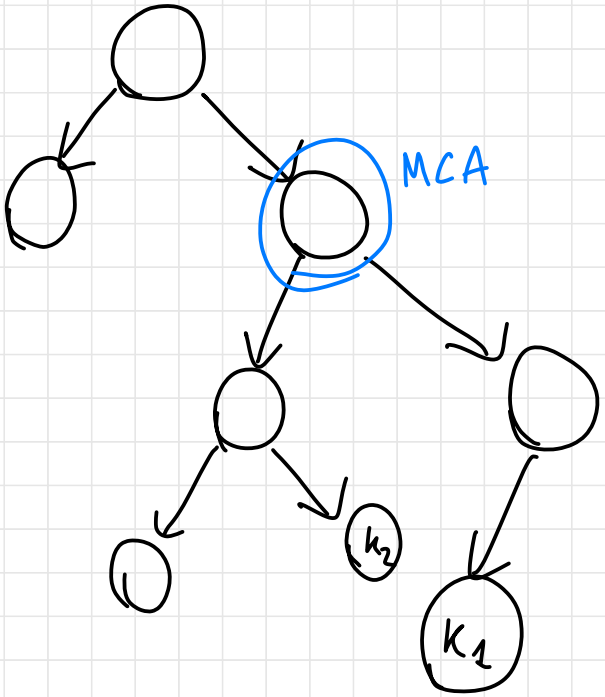
Assunzioni:

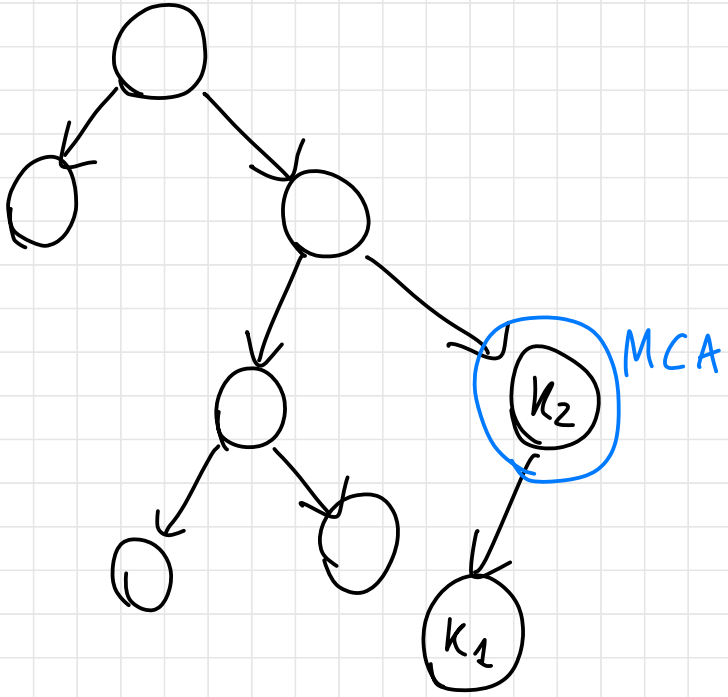
- albero non ha chiavi duplicate
- k_1, k_2 sono presenti nell'albero

Def.: MCA è un antenato comune ad entrambi i nodi più lontano possibile dalla radice

Caso particolare



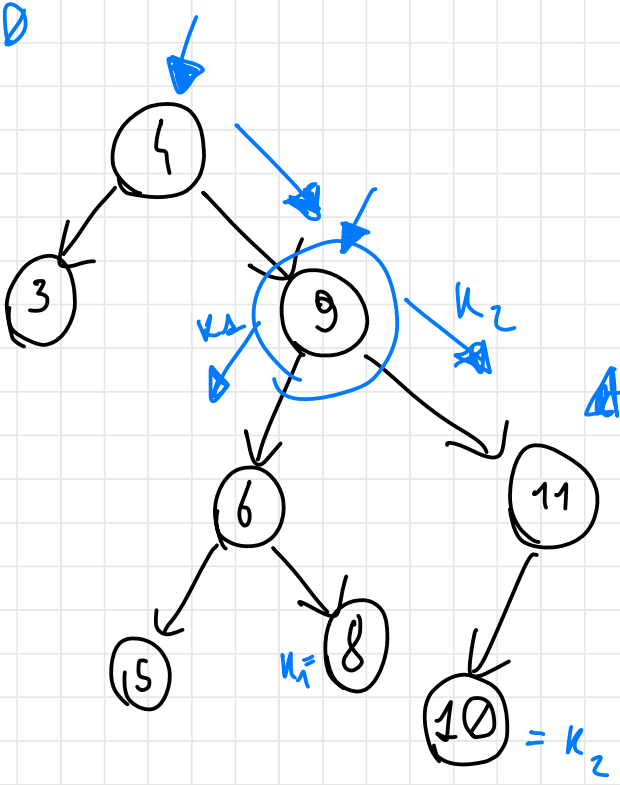

MCA
k_2
k_1


MCA
k_2
k_1

$k_1 < k_2$

8 e 10

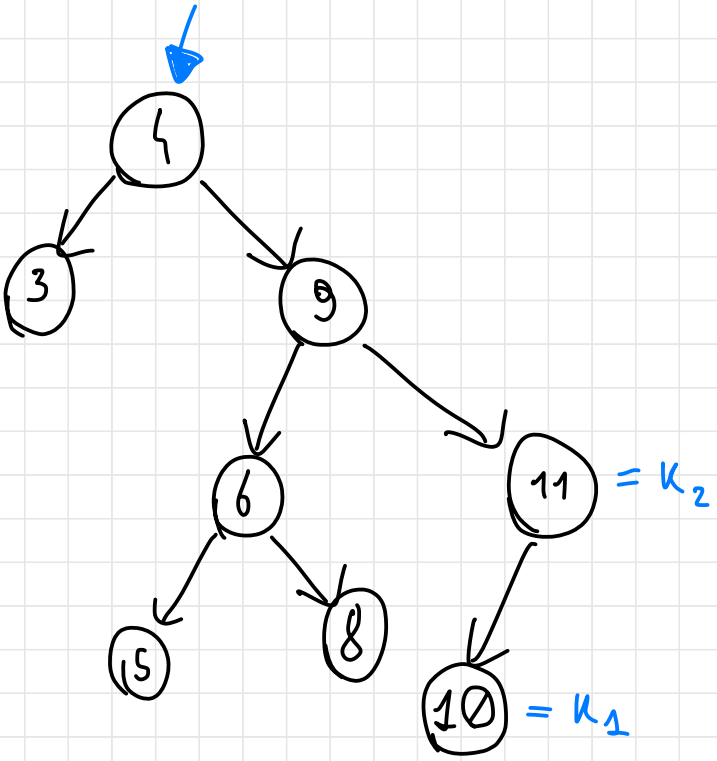
min: il più in basso



da qui in poi non ci saranno antenati comuni

Sol. di Marco : $O(h)$

se il BST è bilanciato $O(\log(n))$



if $k_1 <$ curr.key and $k_2 <$ curr.key

vai a sx

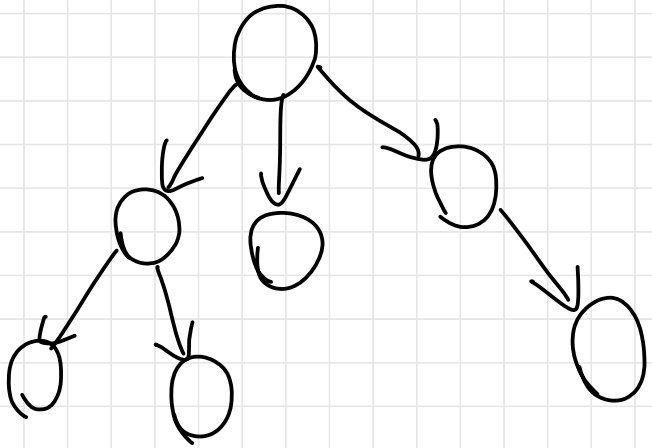
else if $k_1 >$ curr.key and $k_2 >$ curr.key

vai a dx

else if $k_1 \leq$ curr.key $\leq k_2$

ritorna curr

ls: Progettazione albero non binario di ricerca



Ogni nodo ha questa struttura:

n.key

n.lc : puntatore figlio più a sx

n.rs : punt. primo fratello a dx

n.ls : punt. primo fratello a sx

n.p : punt. padre

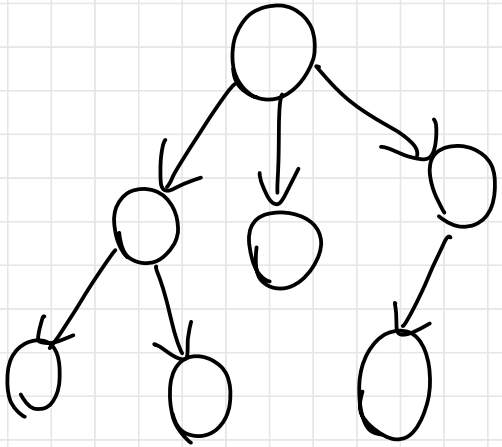
n.key

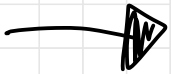
n.lc : puntatore figlio più a sx

n.rs : punt. primo fratello a dx

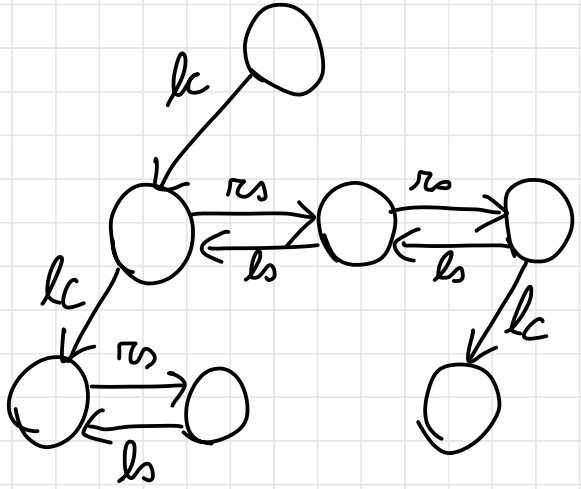
n.ls : punt. primo fratello a sx

n.p : punt. padre





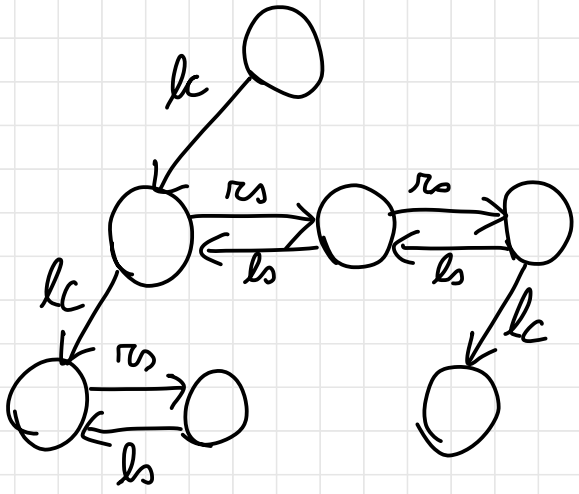
(ho omesso i punt. ai padri)



Richiesta: usare l'albero non binario come se fosse BST

se $k < k'$ vai a sx
$k > k'$ vai a dx

→ progettare l'albero non-bin per avere cond. di ricerca efficienti

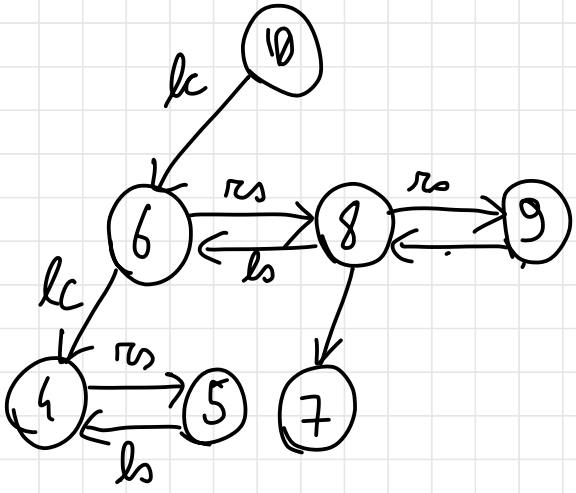


Progetto l'albero così:

Dato un nodo n:

tutti i nodi del sottoalb. con radice in n.lc avranno $k \leq n.k$

[...] con radice in n.rs avranno $k \geq n.k$

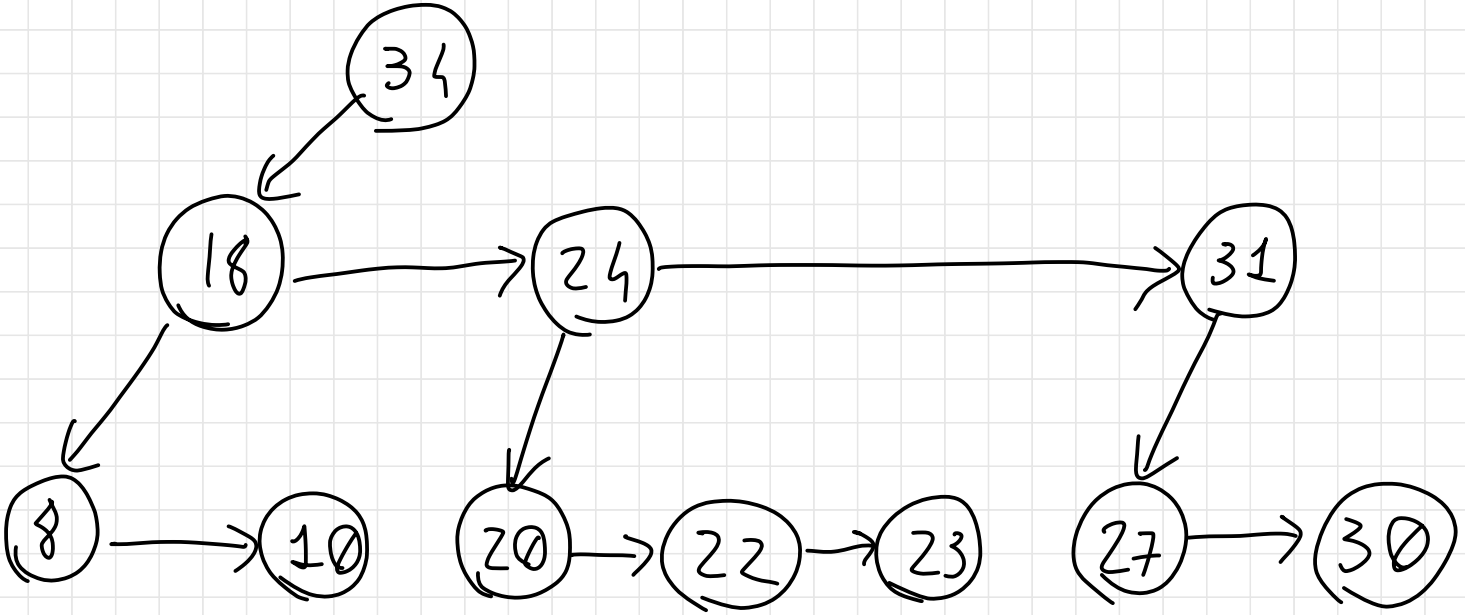


Search: pseudo codice sulle slide

min? scendere più a sx possibile seguendo lc

max? Radice

Succ(24)? Prendo il sottoalbero rs di 24 e trovo il min.
se 24 ha rs



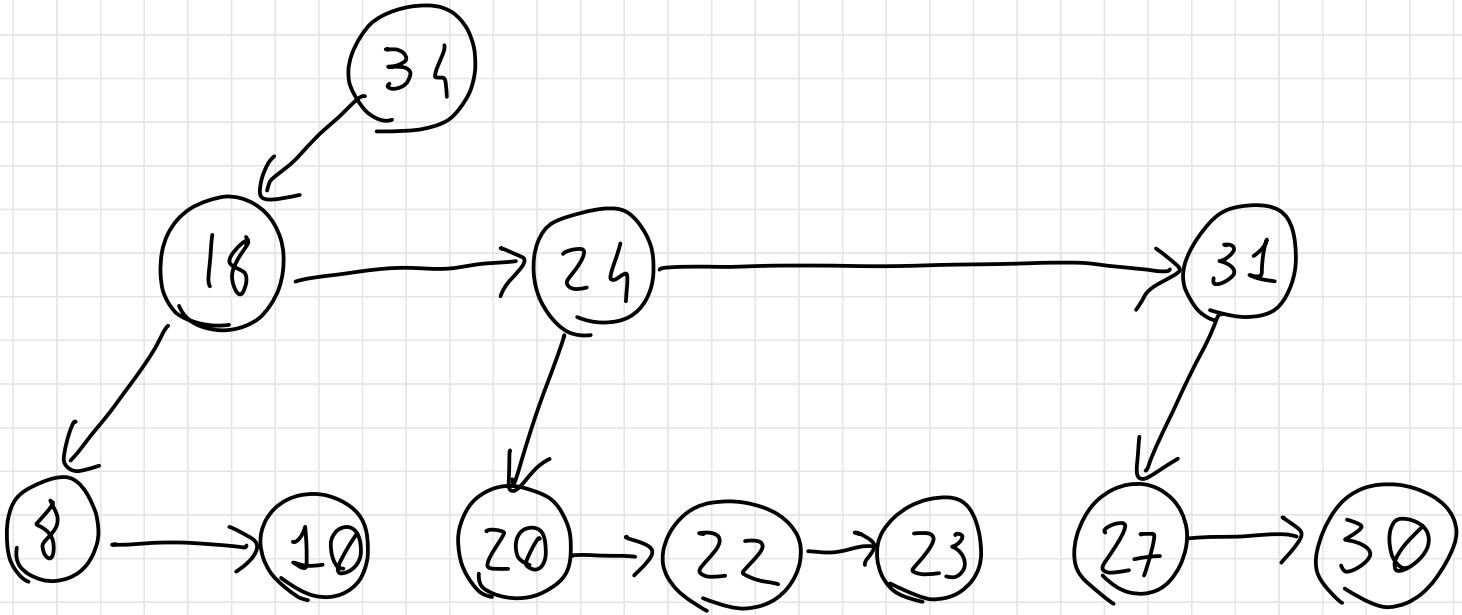
Succ(24)? Prendo il sottoalbero rs di 24 e trovo il min.
se 24 ha rs

Se 24 non ha rs, il succ. sarà il padre

Pred(24)? Prendo sottoalbero in l_c, prendo il max se ha in l_c

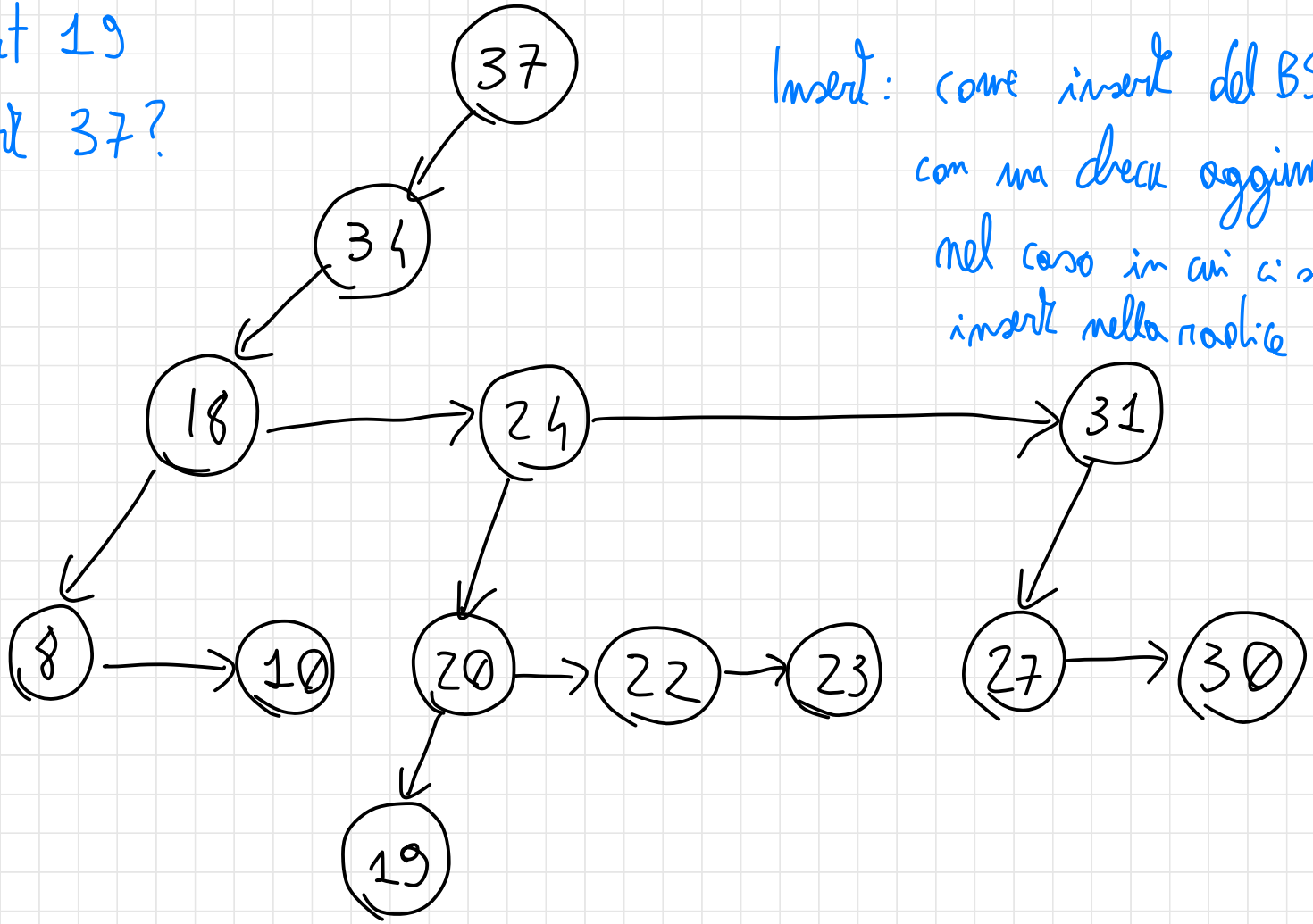
Se non ha l_c? l_s

Se non ha l_c e nemmeno l_s? l_s del padre



Insert 19

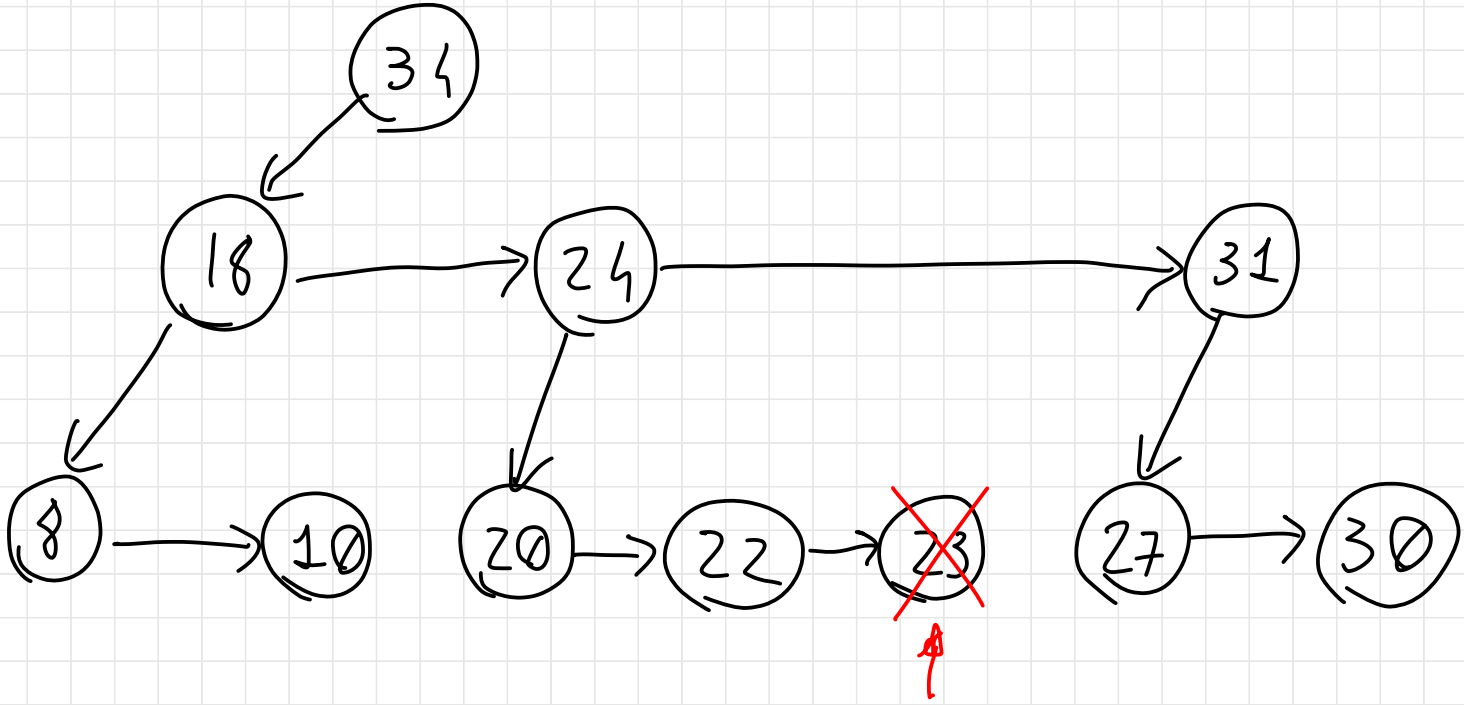
Insert 37?



Insert: come insert del BST con una check aggiuntivo nel caso in cui ci sia insert nella radice

Delete: Caso 1: elimino nodo senza figli o fratelli

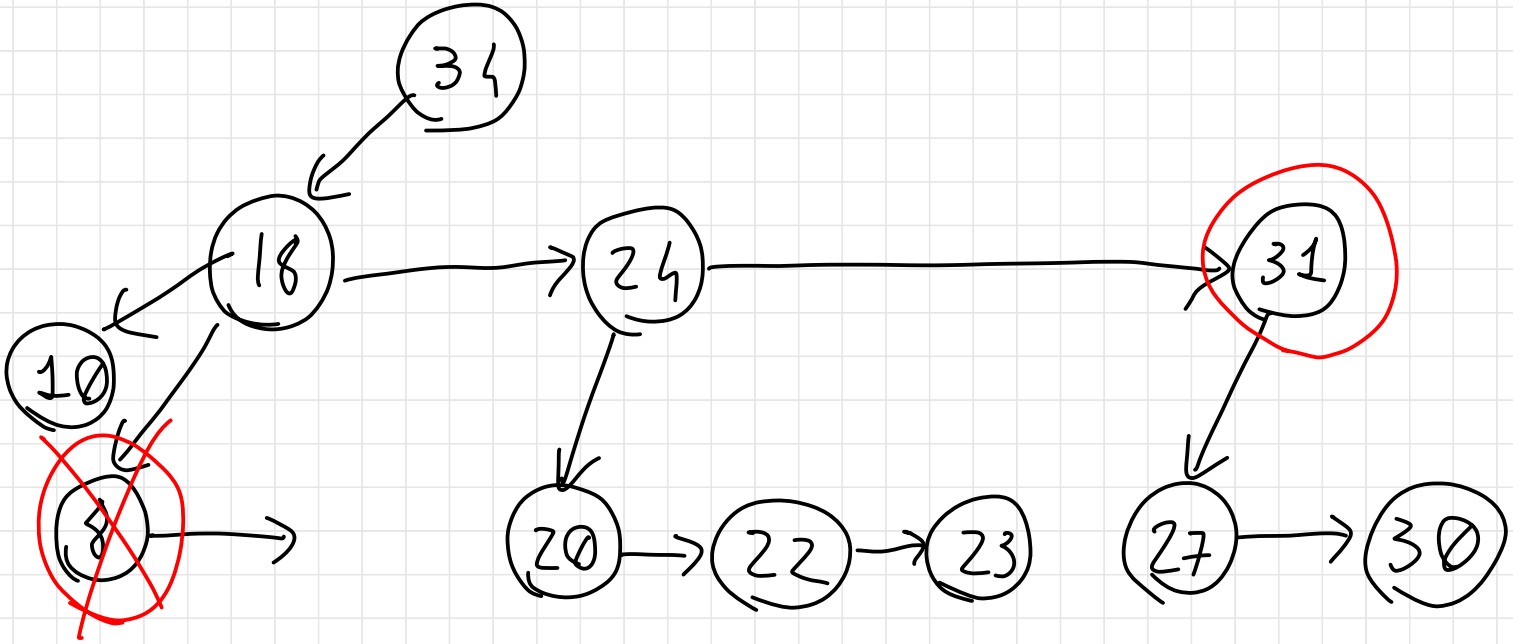
→ elimino il nodo senza problemi

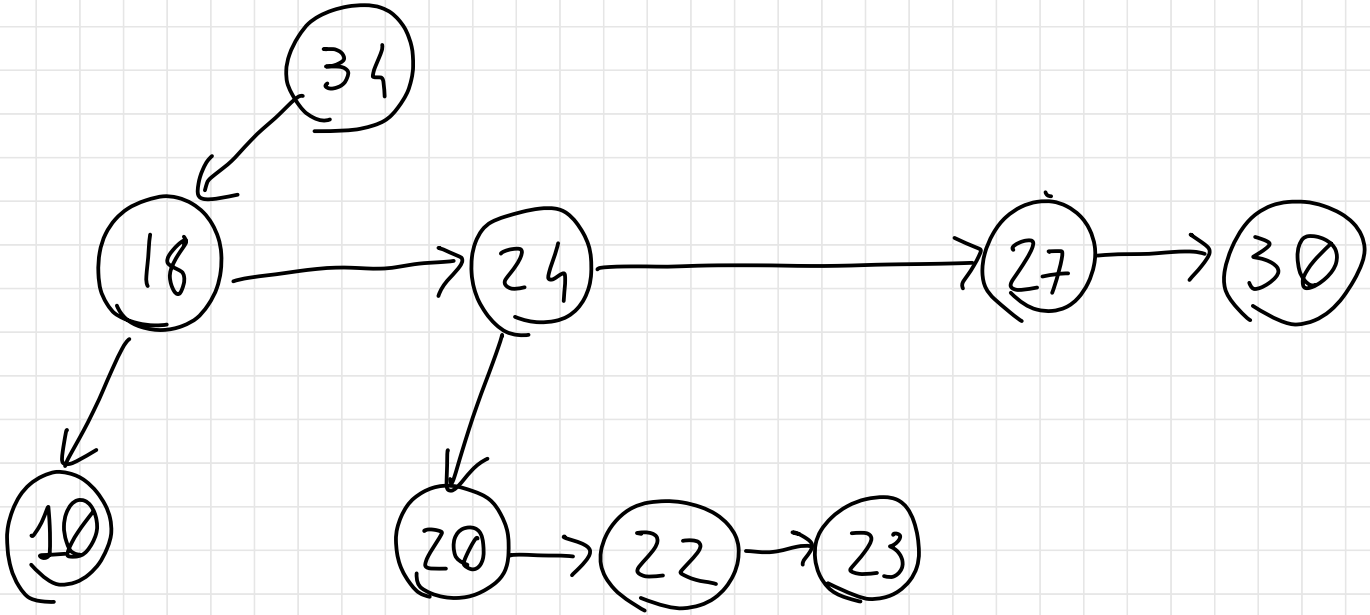


Delete: Caso 2: nodo con rs ma senza figli

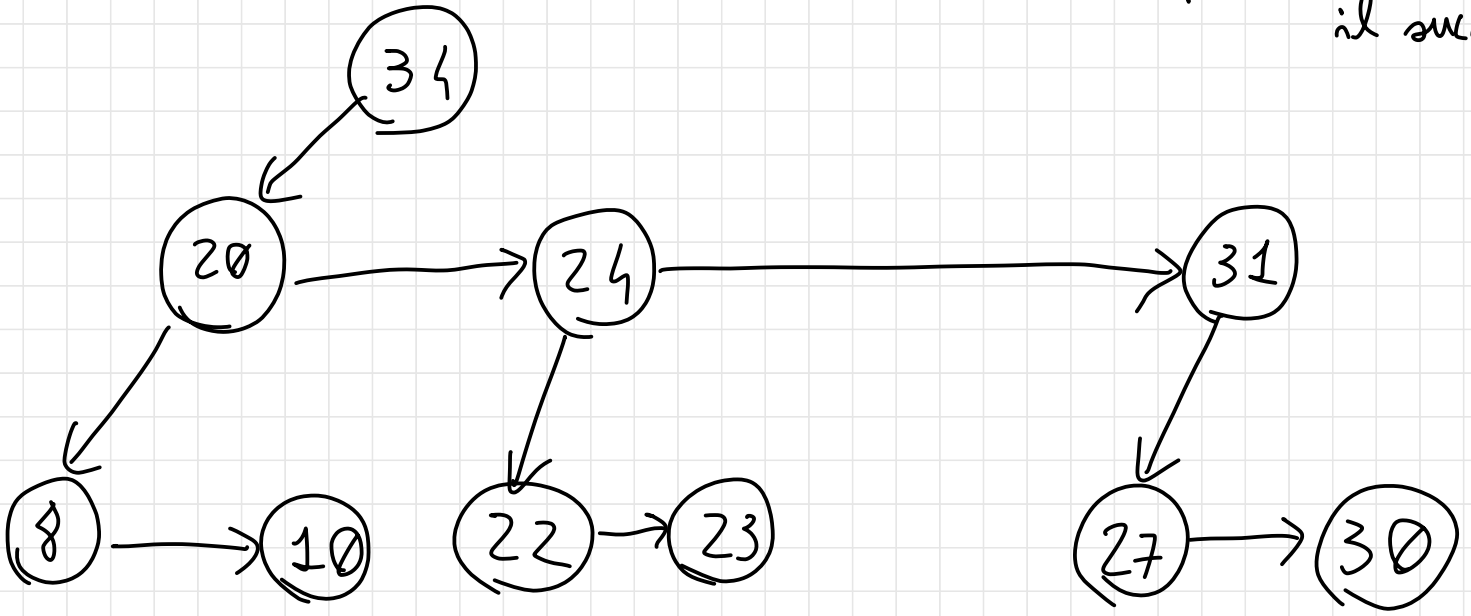
copio nel nodo con rs e elimino rs

simile quando il nodo ha lc ma non ha rs




34
18
24
27
30
10
20
22
23

Delete Caso 3: sia rs che lc: il succ. prende il posto del nodo, poi si elimina il succ



Delete Caso 4: elimino la radice → cerco il pred. della radice, copio il pred. nella radice, elimino il pred.