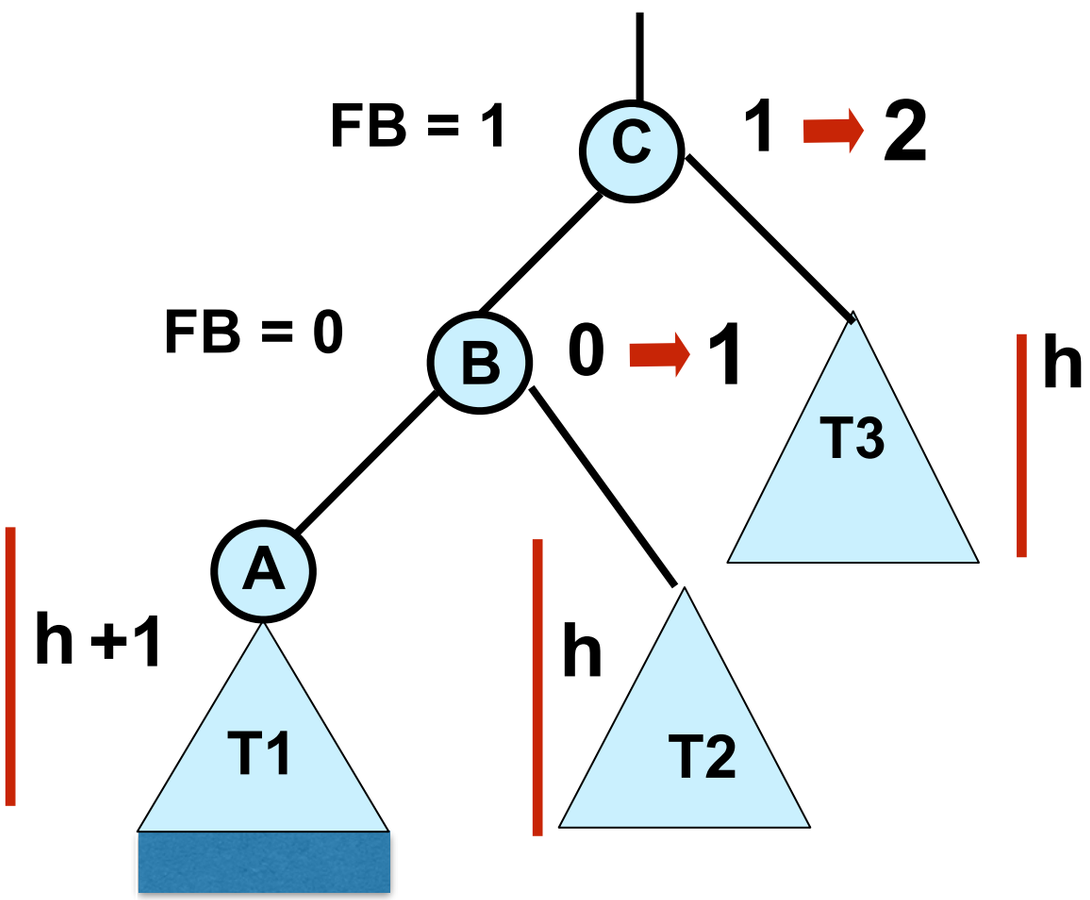
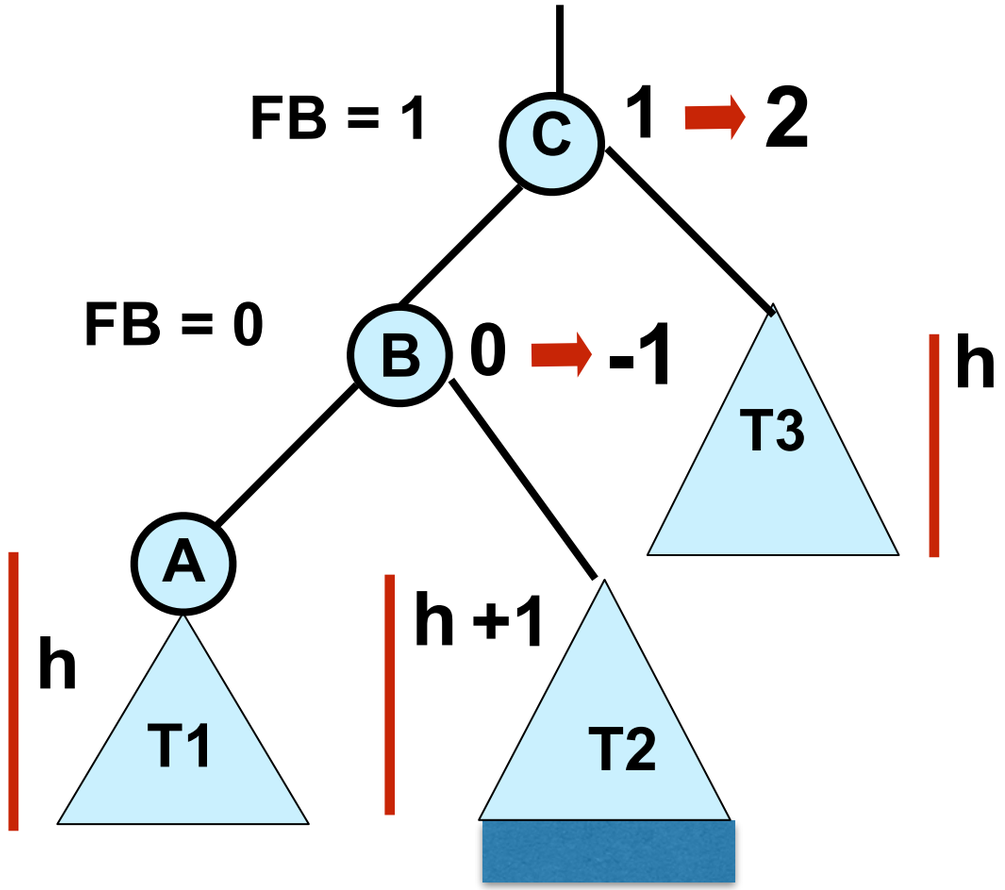
Cenni sulla cancellazione in un AVL esercizi su rotazioni e relazioni di ricorrenza

Casi di sbilanciamento inserimento





left- left

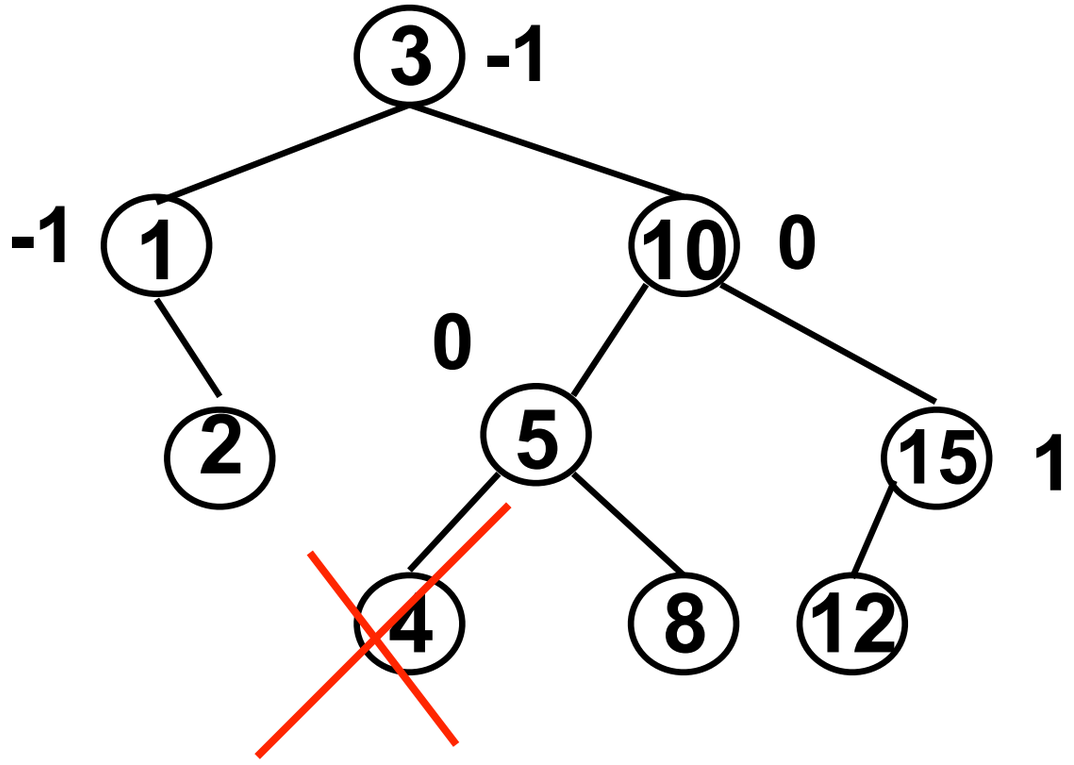
left- right

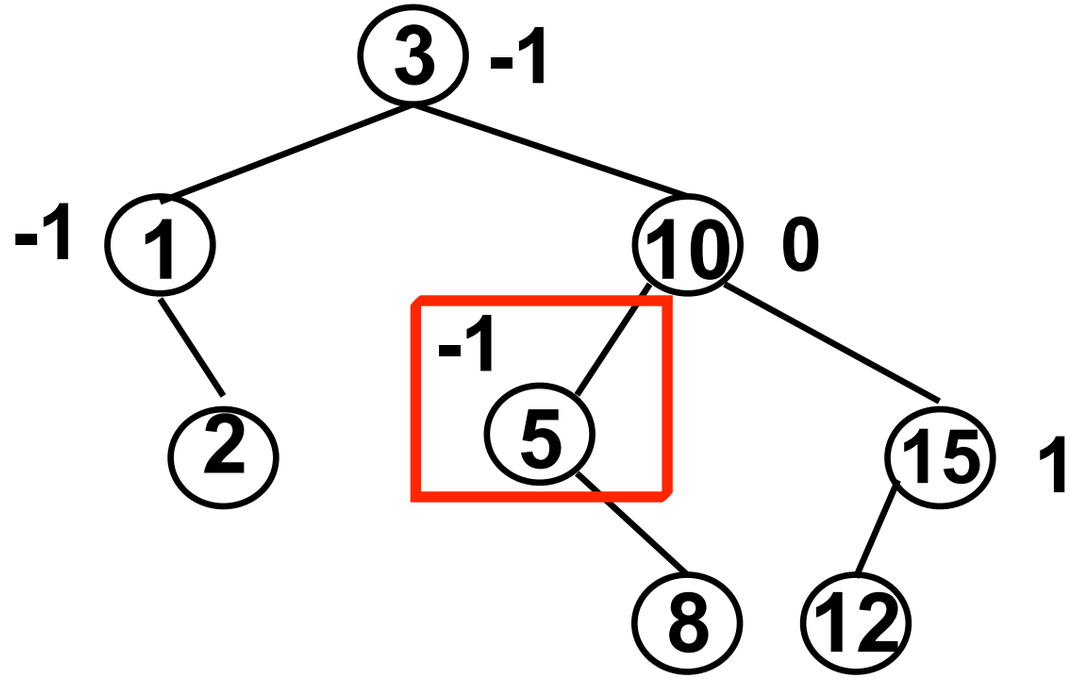
A questi si devono aggiungere i simmetrici: right-right, right-left

La cancellazione in un AVL

La cancellazione avviene come in un ABR, con la necessità di aggiornare i fattori di bilanciamento ed eventualmente ribilanciare l'albero.

Esempio:
Cancellazione di una foglia, 4

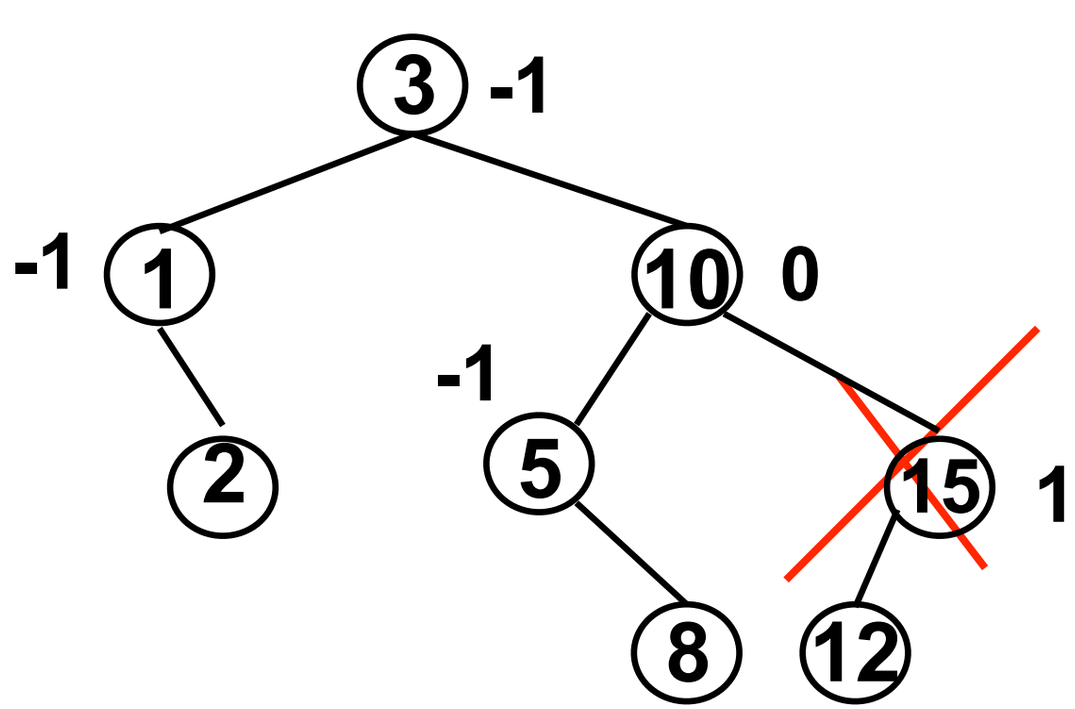


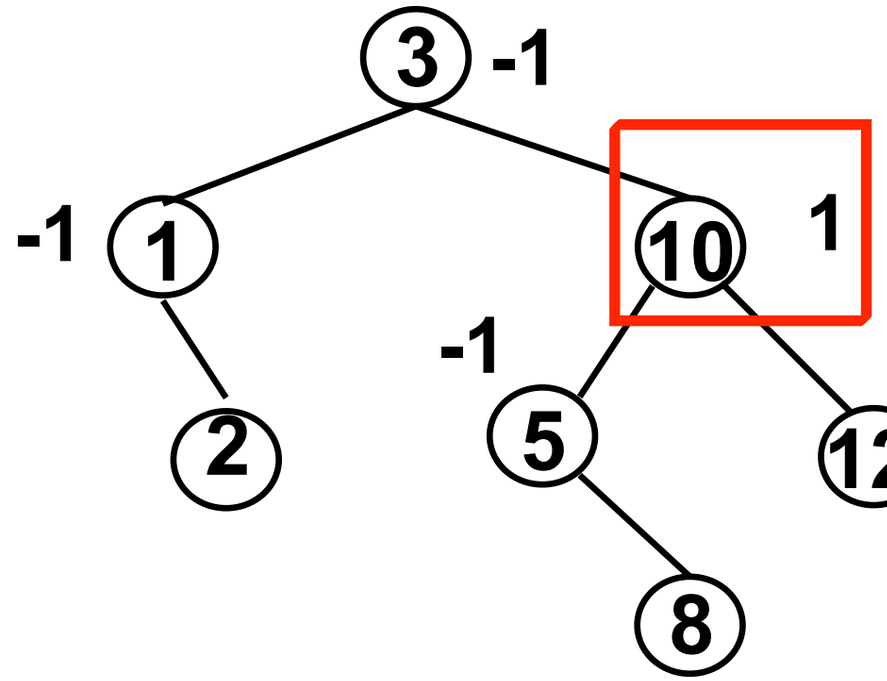


unico FB da aggiornare

La cancellazione in un AVL

Esempio:
Cancellazione di un nodo con un solo figlio, 15

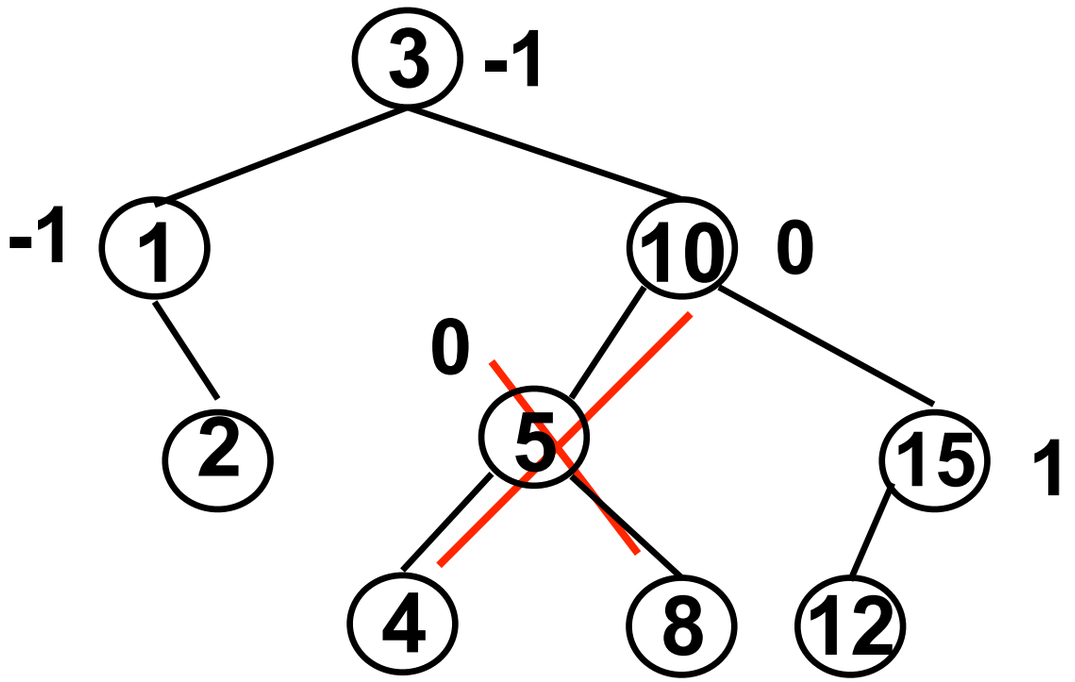


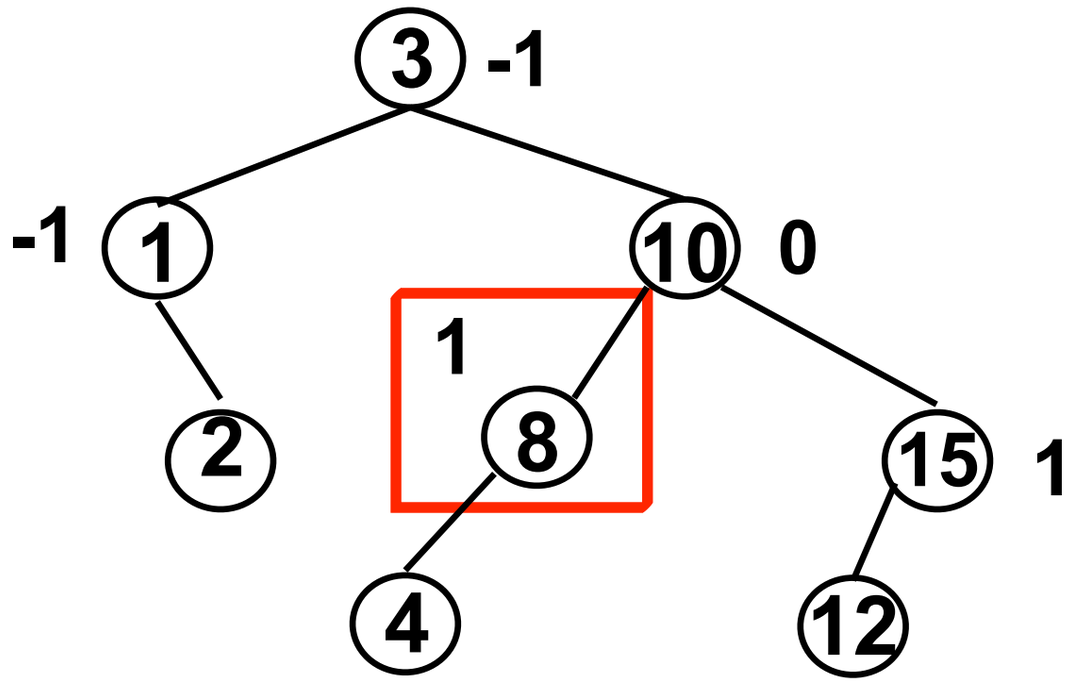


unico FB da aggiornare

La cancellazione in un AVL

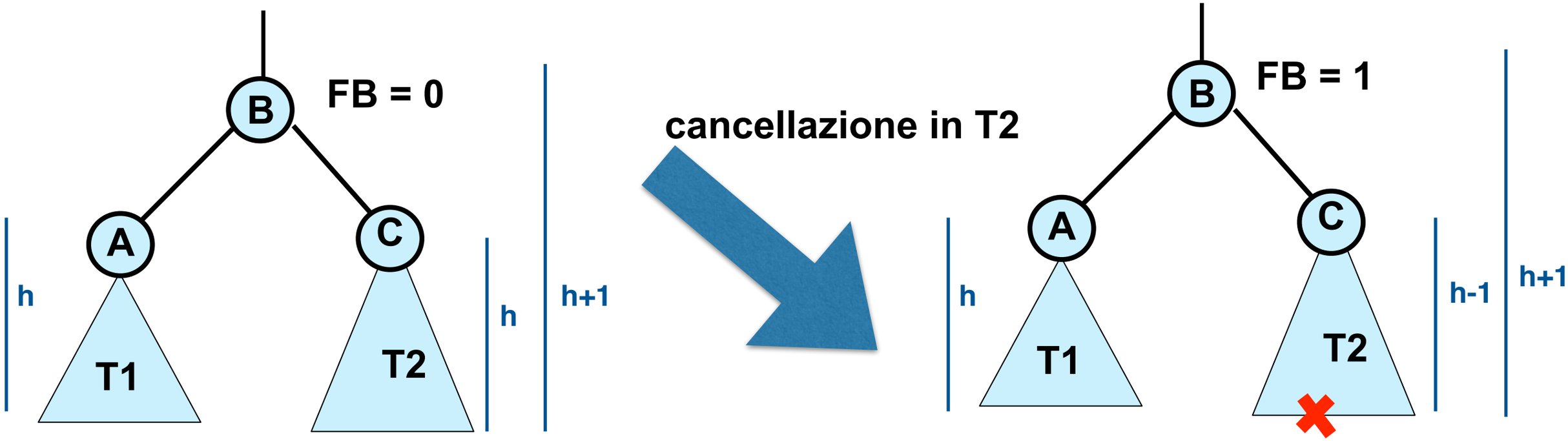
Esempio:
Cancellazione di un nodo con DUE figli, 5





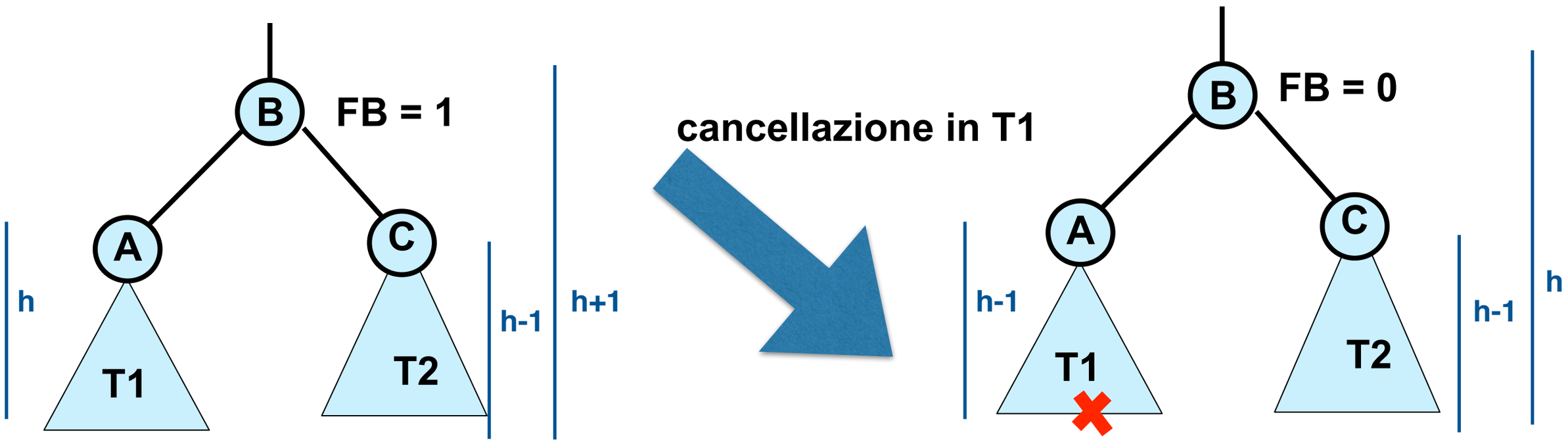
unico FB da aggiornare

La cancellazione in un AVL: aggiornamento FB



Se risalendo verso la radice il fattore di bilanciamento passa da 0 a 1, o da 0 a -1, si può fermare il processo di aggiornamento dei fattori di bilanciamento, perché vuol dire che la cancellazione non ha prodotto una diminuzione di altezza di tutto l'albero ma solo in un suo sotto albero.

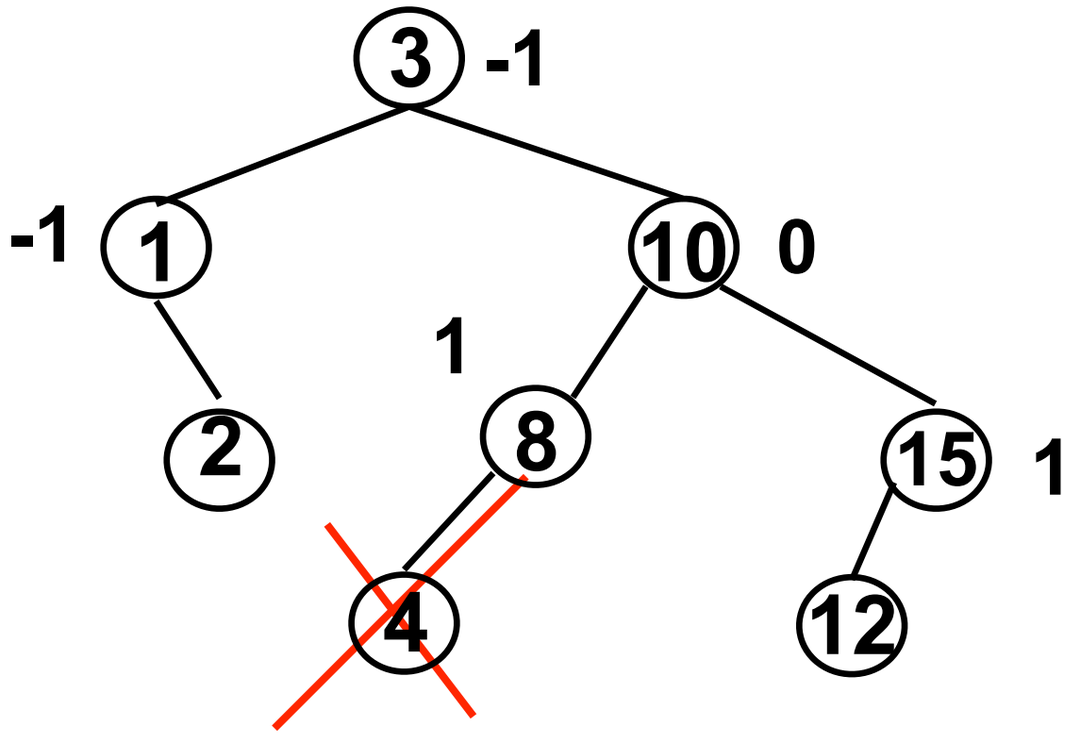
La cancellazione in un AVL: aggiornamento FB

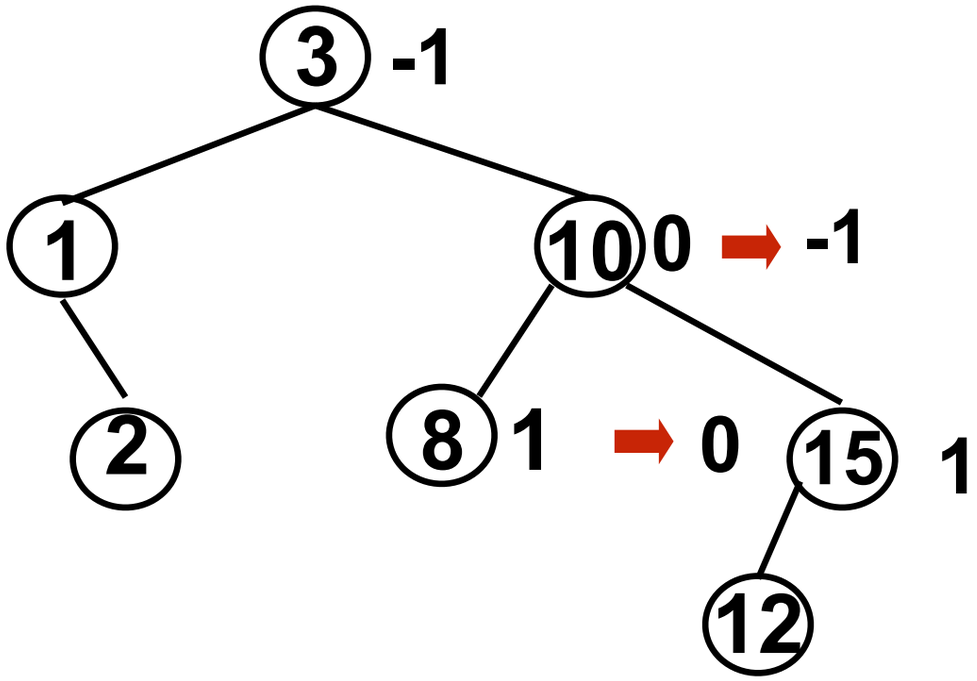


Se risalendo verso la radice il fattore di bilanciamento passa da 1 a 0, o da -1 a 0, non si può fermare a B il processo di aggiornamento dei fattori di bilanciamento, perché vuol dire che la cancellazione ha prodotto una diminuzione di altezza di tutto l'albero non solo in un suo sotto albero.

La cancellazione in un AVL

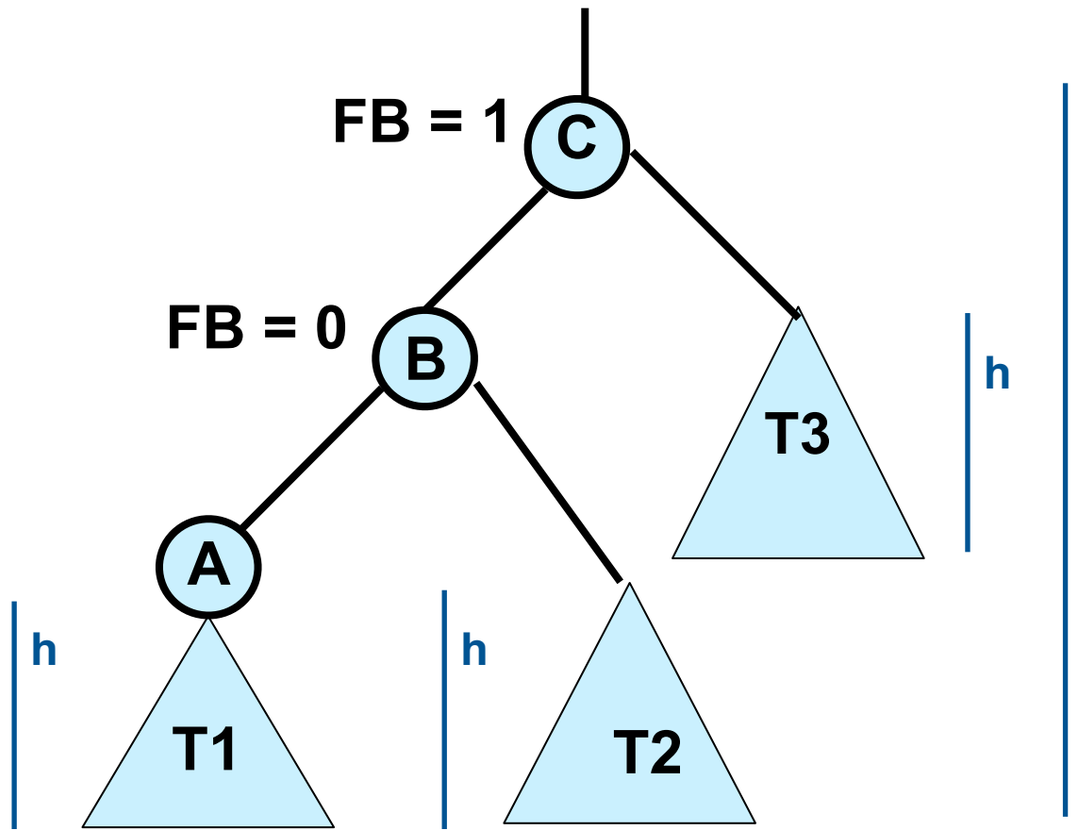
Esempio:
Cancellazione e aggiornamento di più FB





FB da aggiornare

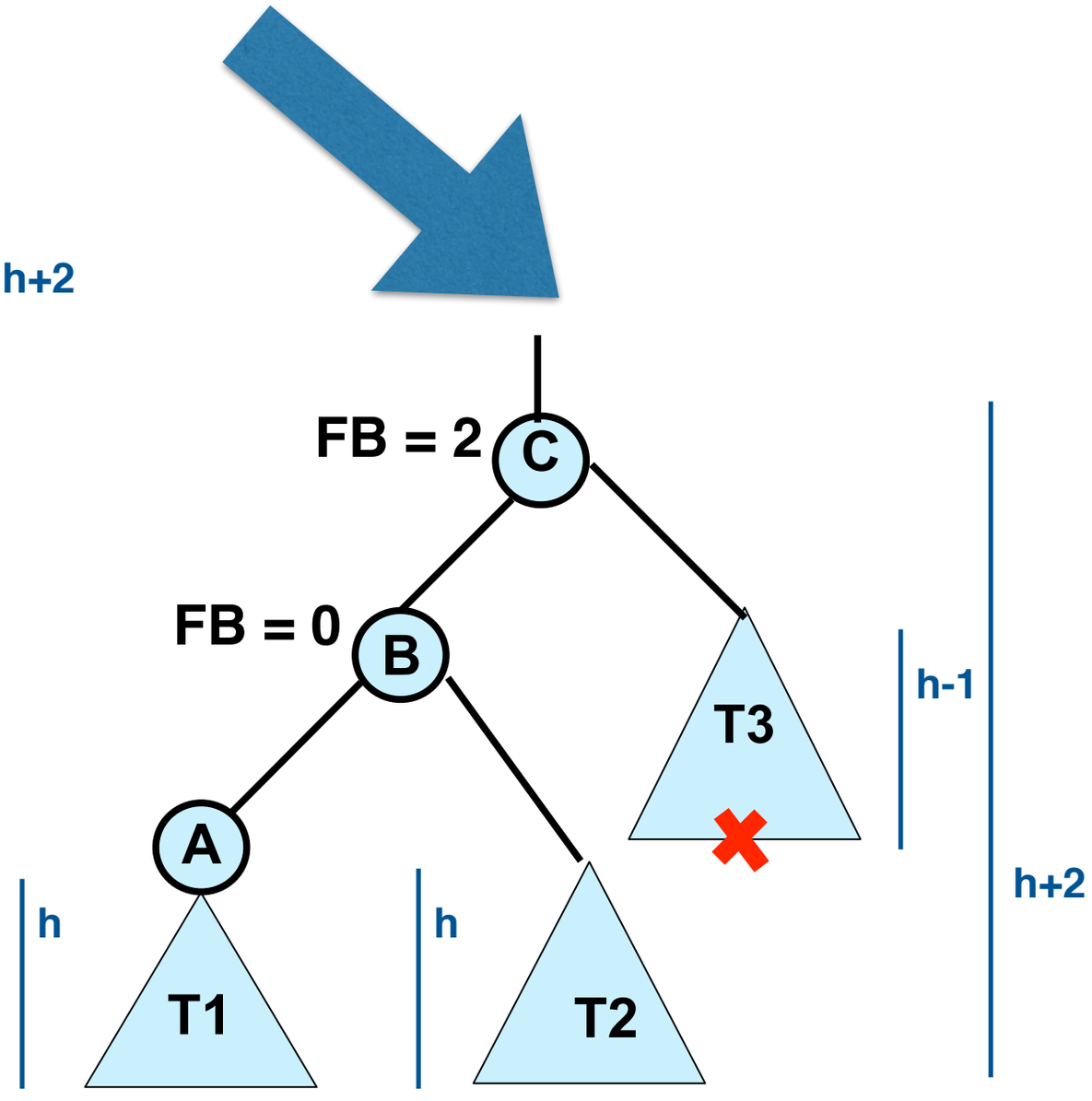
La cancellazione: caso left-left 1



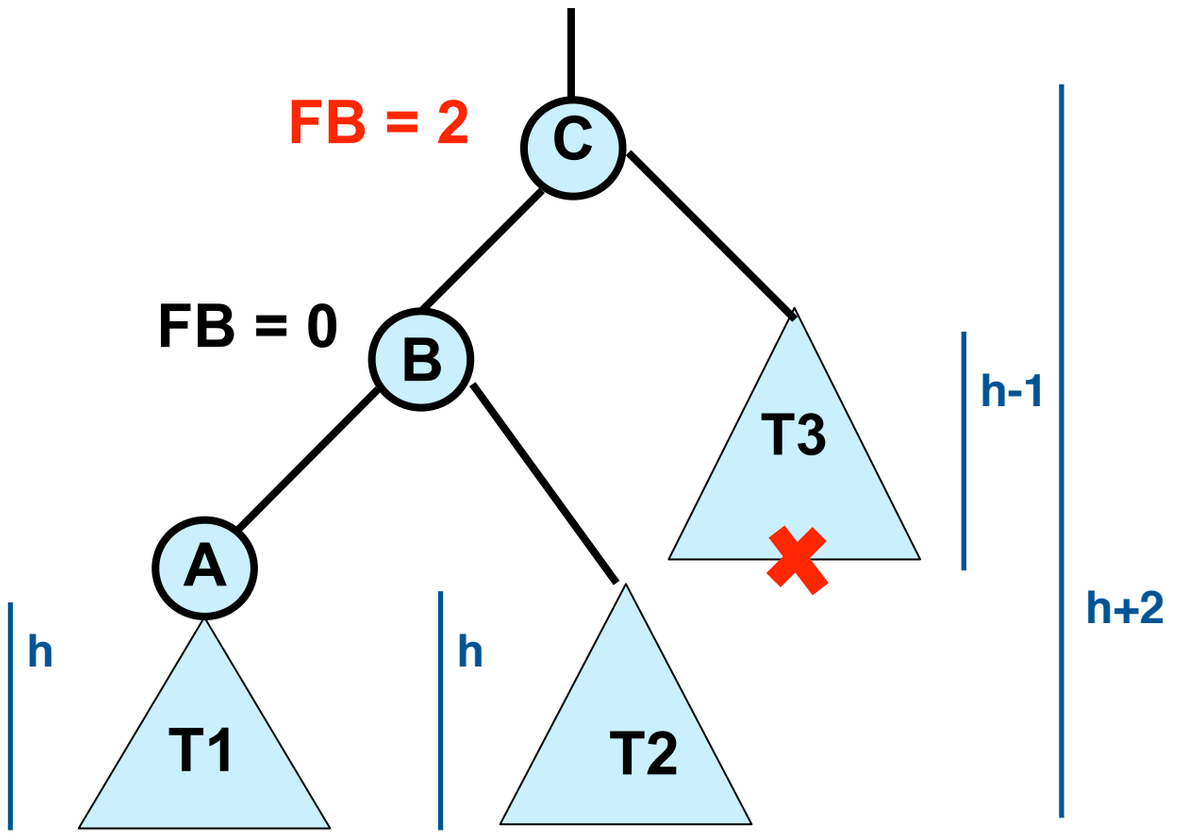
Si chiama LEFT LEFT perché corrisponde ad un inserimento nel sotto albero sinistro del sotto albero sinistro, T1.

cancellazione in T3

La cancellazione in T3 ne provoca una diminuzione di altezza, quindi nel nodo C il fattore di bilanciamento diventa illegale, anche se l'altezza del sotto albero radicato in C non cambia. Risalendo verso la radice da destra si incontra un nodo con FB = 2, mentre il suo figlio sinistro ha FB = 0, questo identifica il caso left-left1.

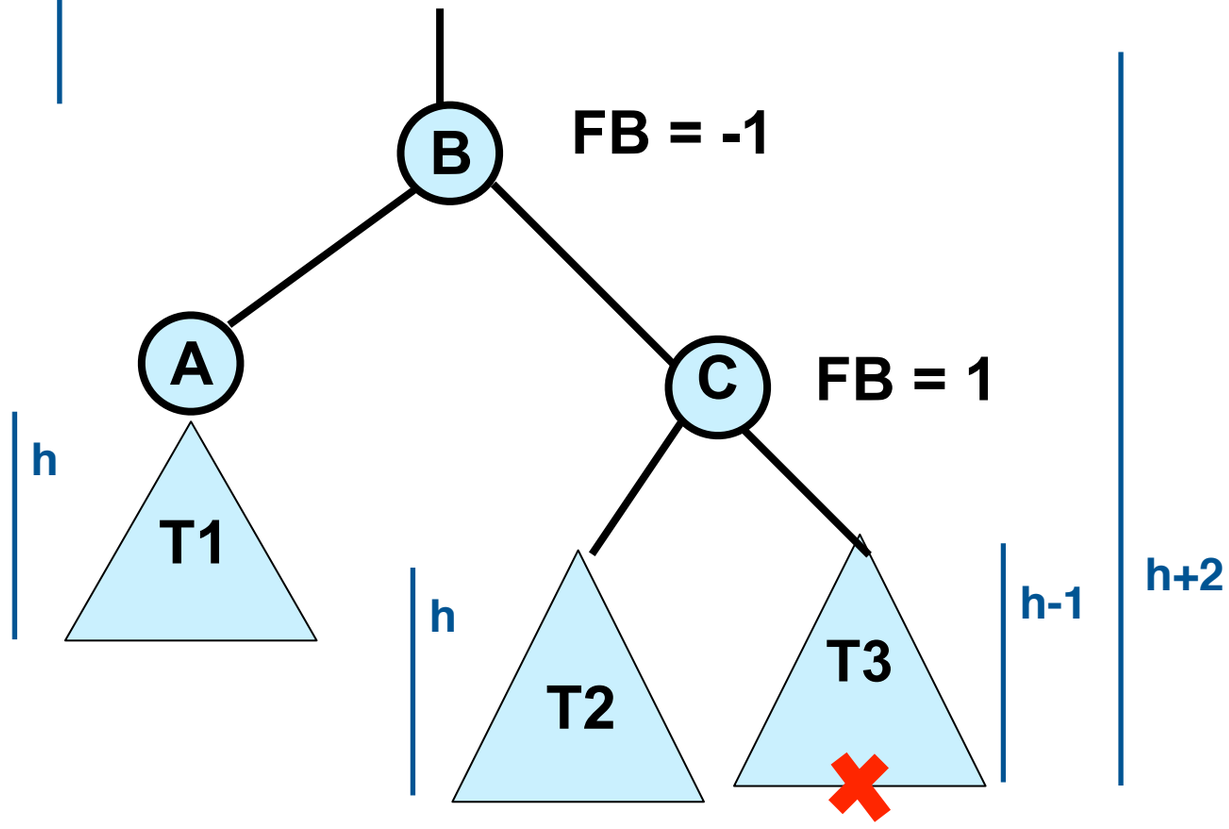


CASO LEFT LEFT 1- rotazione

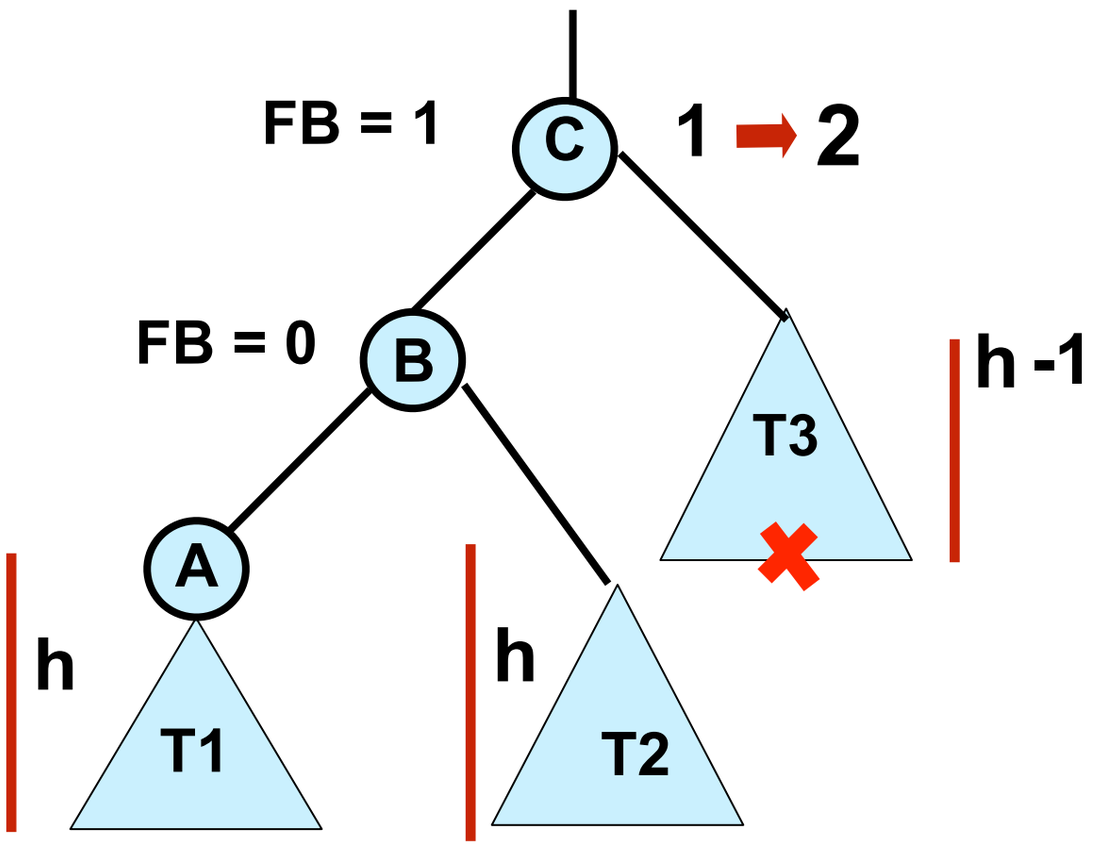


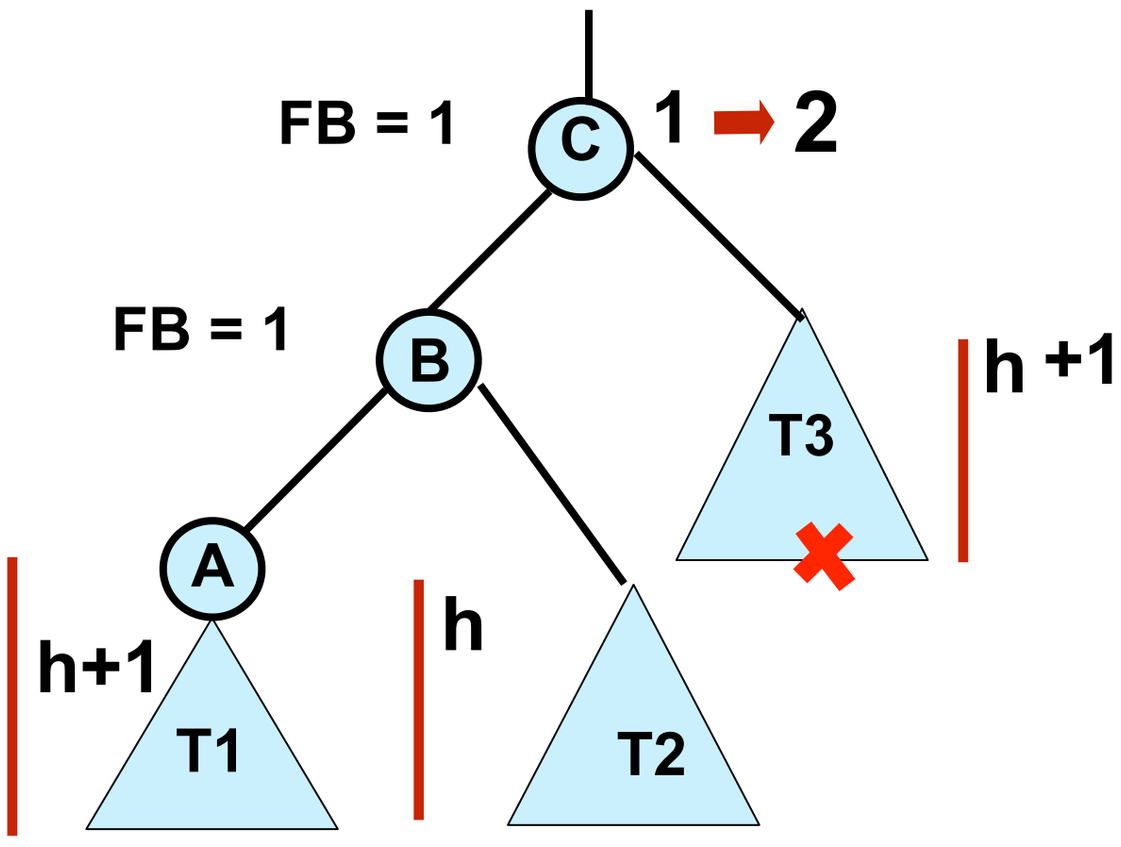
rotazione a destra su C

L'altezza del sottoalbero radicato in B, che rimpiazza quello radicato in C, non è cambiata. Quindi non sono necessarie altre operazioni.
FB = 1 di C diventa -1 per B.



Rotazione a destra



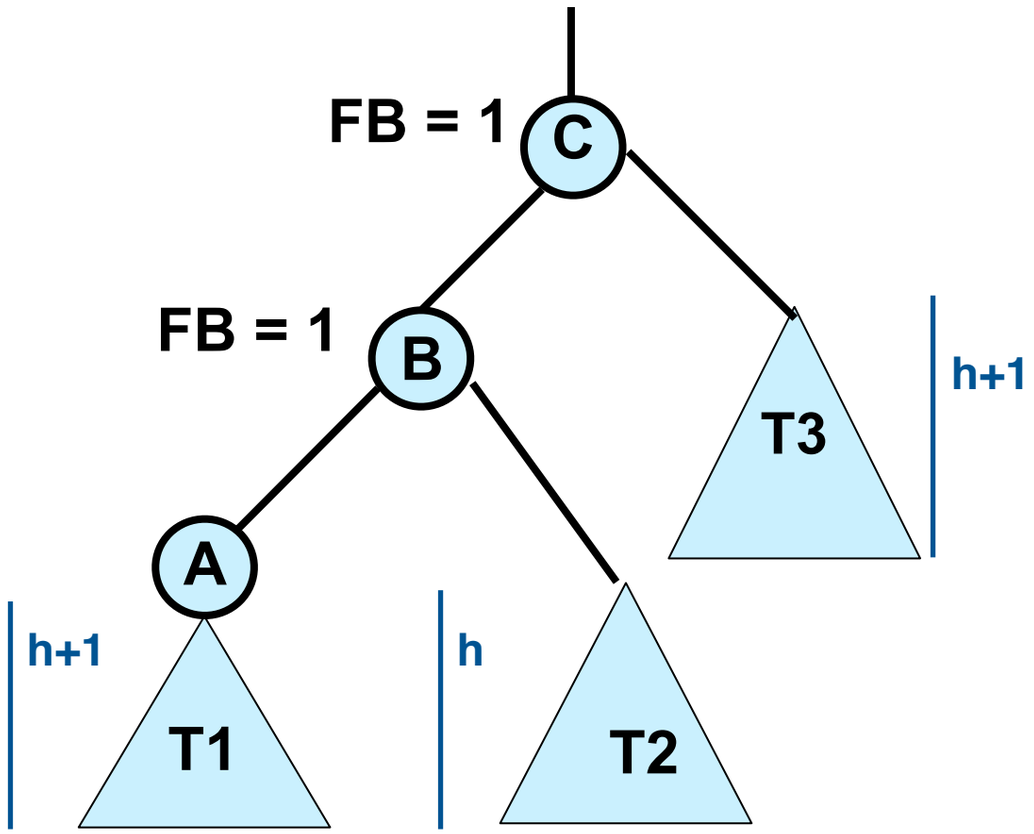


left- left 1

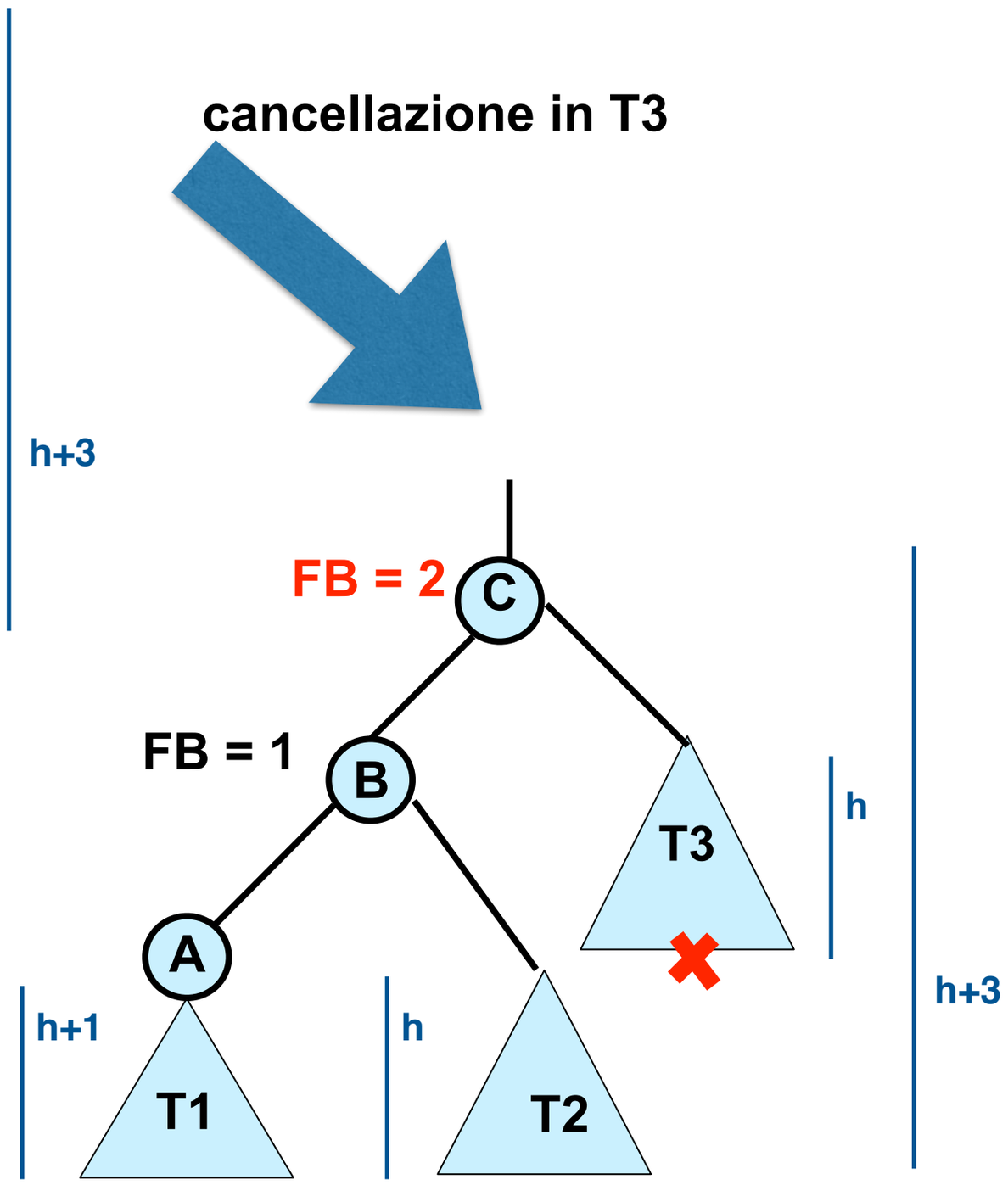
left- left 2

Questi due casi si risolvono con una rotazione a destra, la differenza sta nel fatto che nel primo caso la rotazione è conclusiva, nel secondo l'altezza del sotto albero radicato in B che rimpiazza quello radicato in C ha un'altezza minore di 1 rispetto al sotto albero radicato in C prima della cancellazione e quindi si potrebbero dover eseguire altre rotazioni. A questi si devono aggiungere i simmetrici: right-right 1, right-right 2

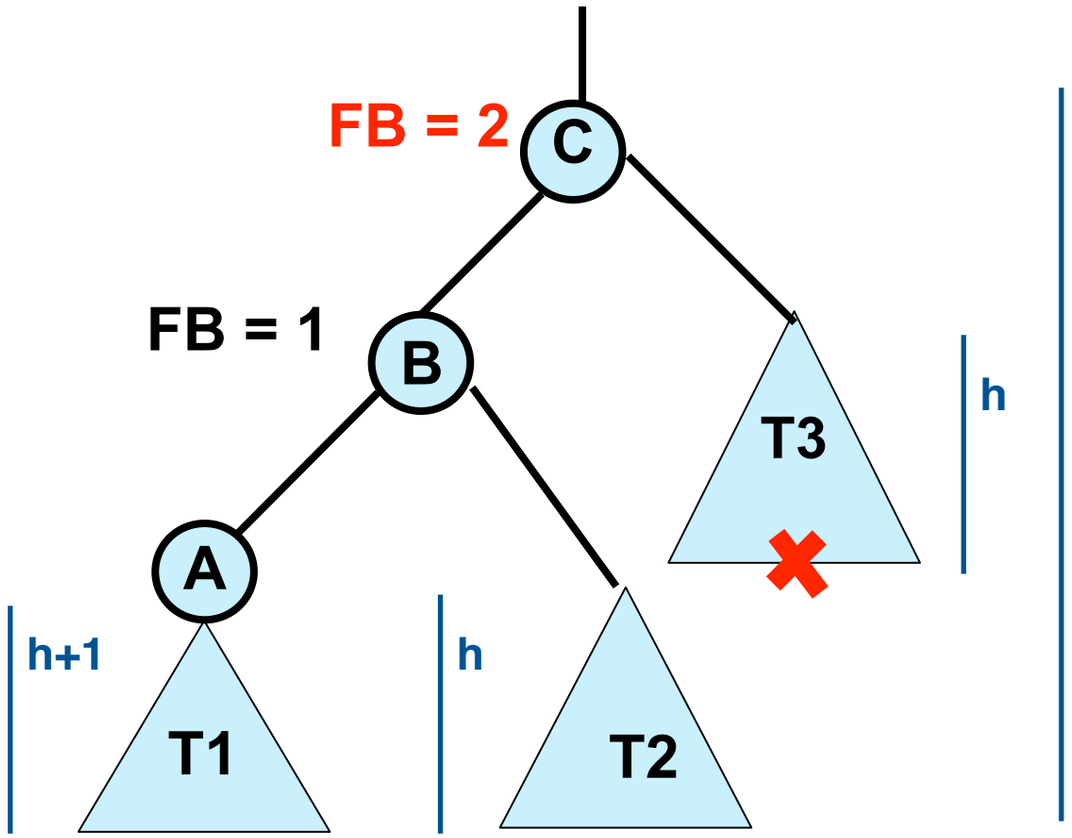
CASO LEFT LEFT 2



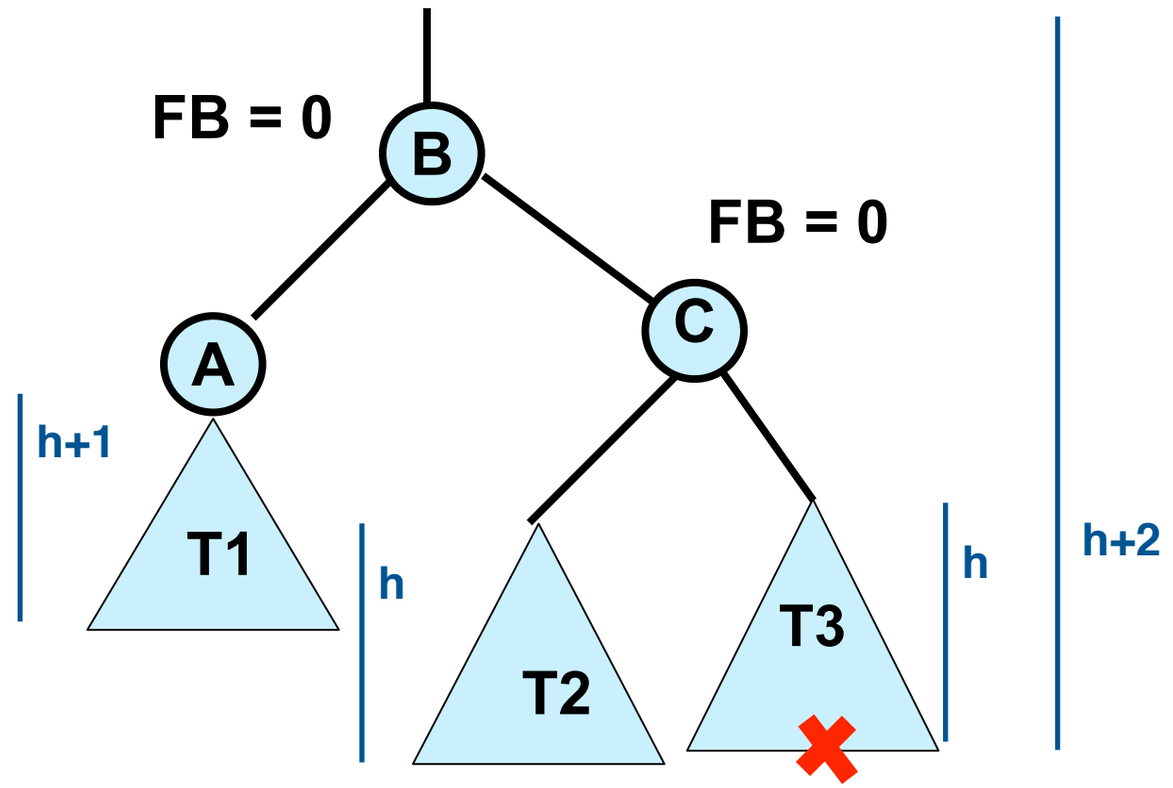
Se risalendo verso la radice da destra si incontra un nodo con FB = 2, il cui figlio sinistro ha FB = 1, ci si trova nel caso left-left2.



CASO LEFT LEFT 2

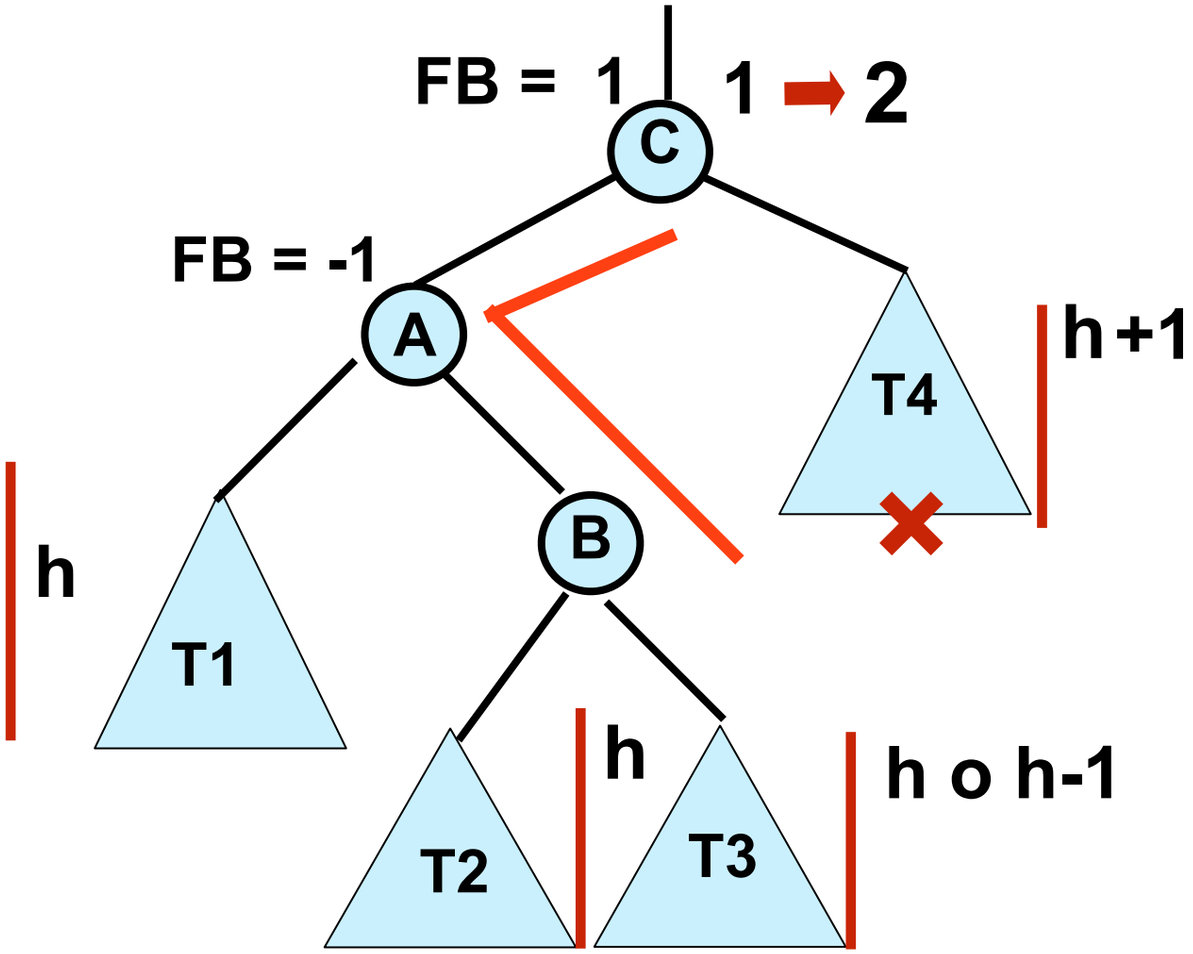


rotazione a destra su C



<u>La cancellazione e la rotazione hanno prodotto una diminuzione di altezza</u> del sotto albero radicato in B, che ha sostituito quello radicato in C. Bisogna risalire al padre di B, se non è la radice, per modificare i fattori di bilanciamento ed eventualmente ribilanciare con altre rotazioni.

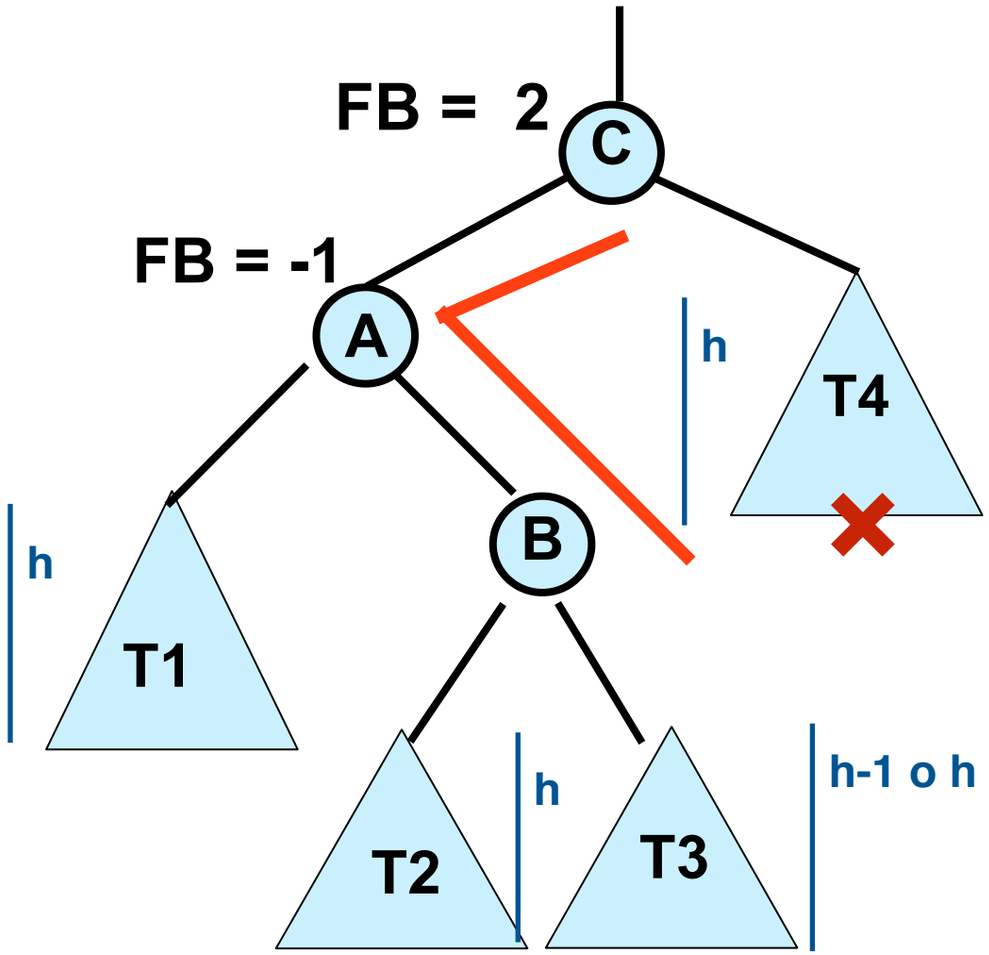
Doppia rotazione



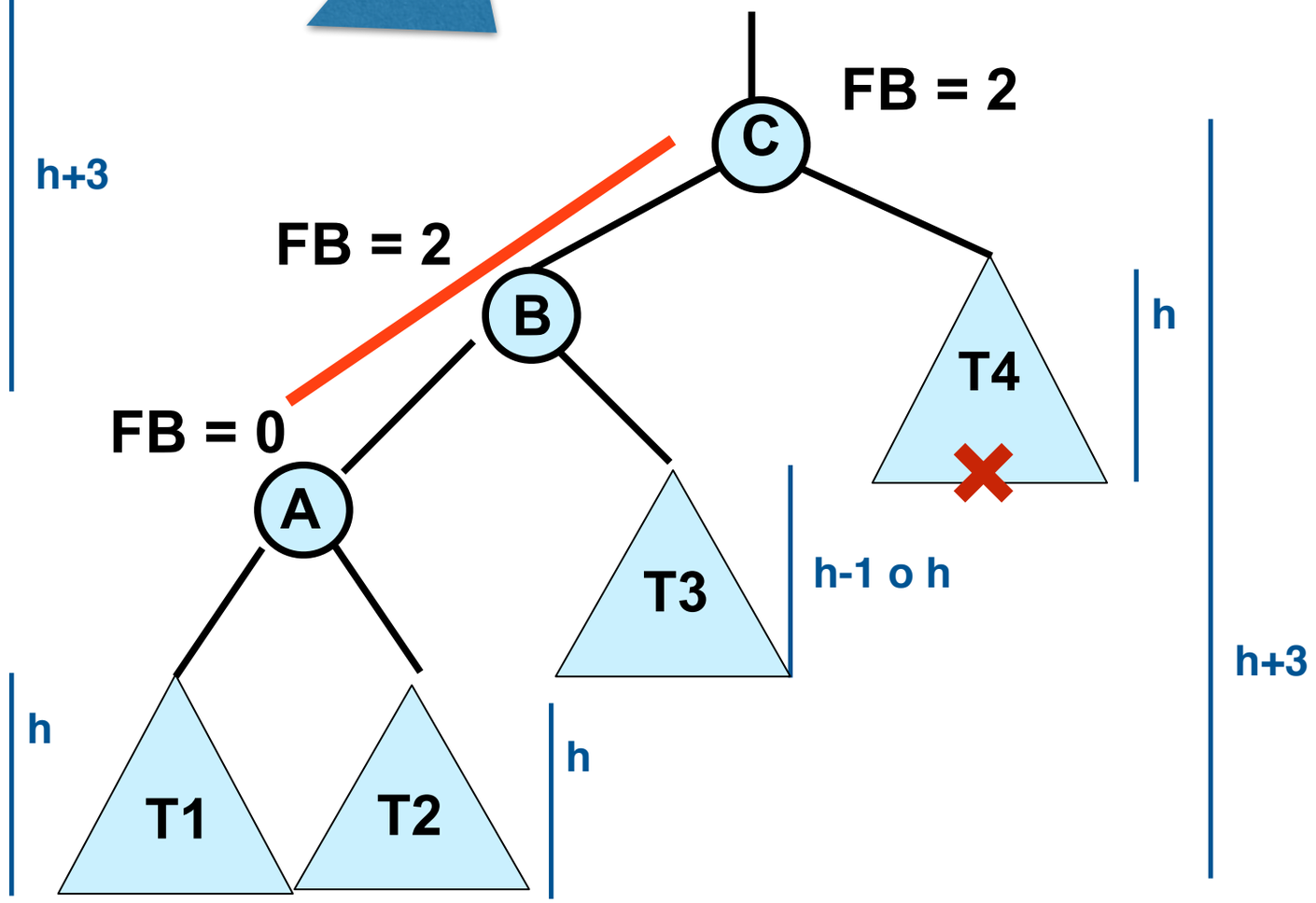
Questo caso si risolve con una rotazione a sinistra su A e una rotazione a destra su C. Anche in questo caso l'altezza del sotto albero radicato in B che sostituirà quello radicato in C sarà di 1 meno alto e quindi possono rendersi necessarie altre rotazioni.

left- right

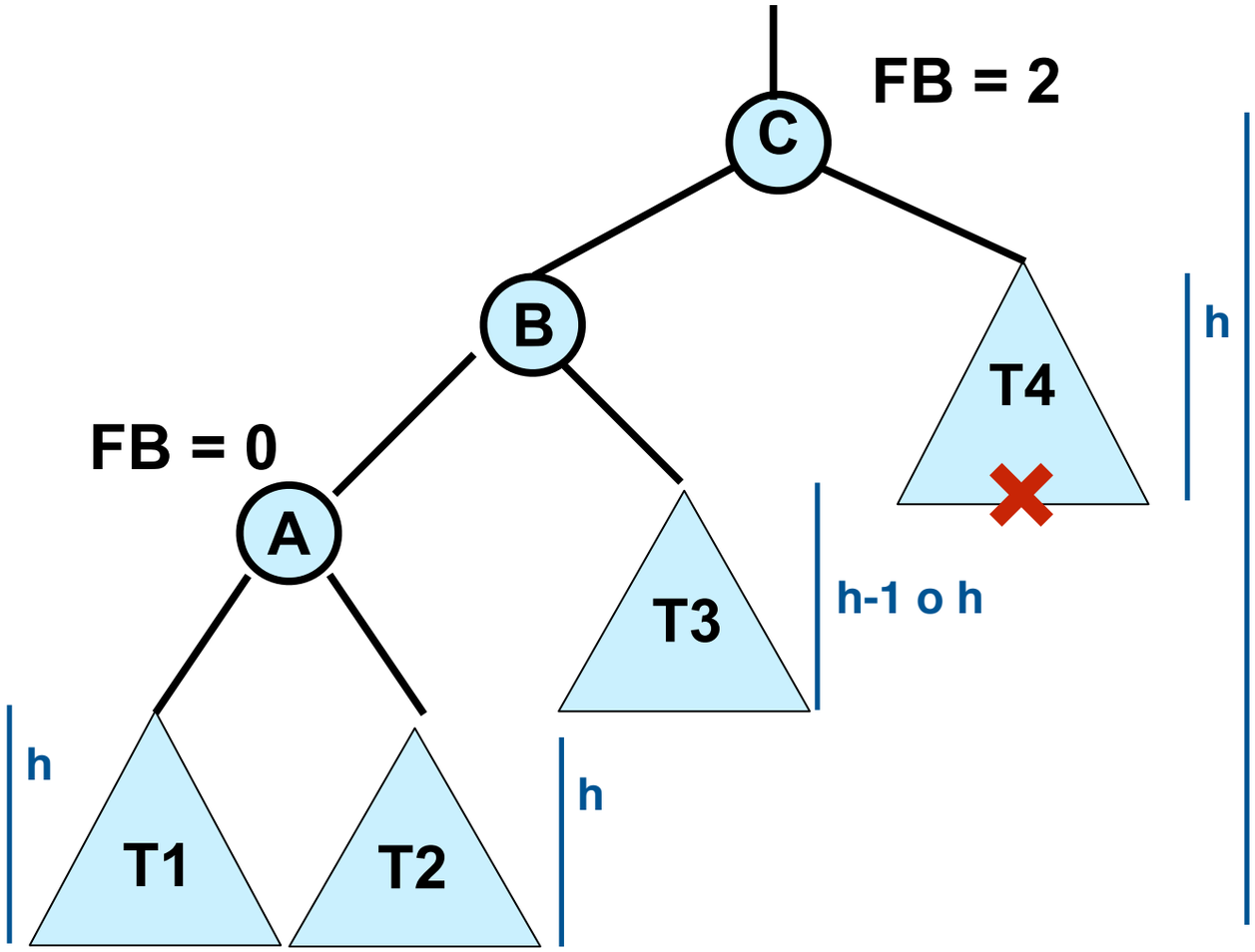
CASO LEFT-RIGHT



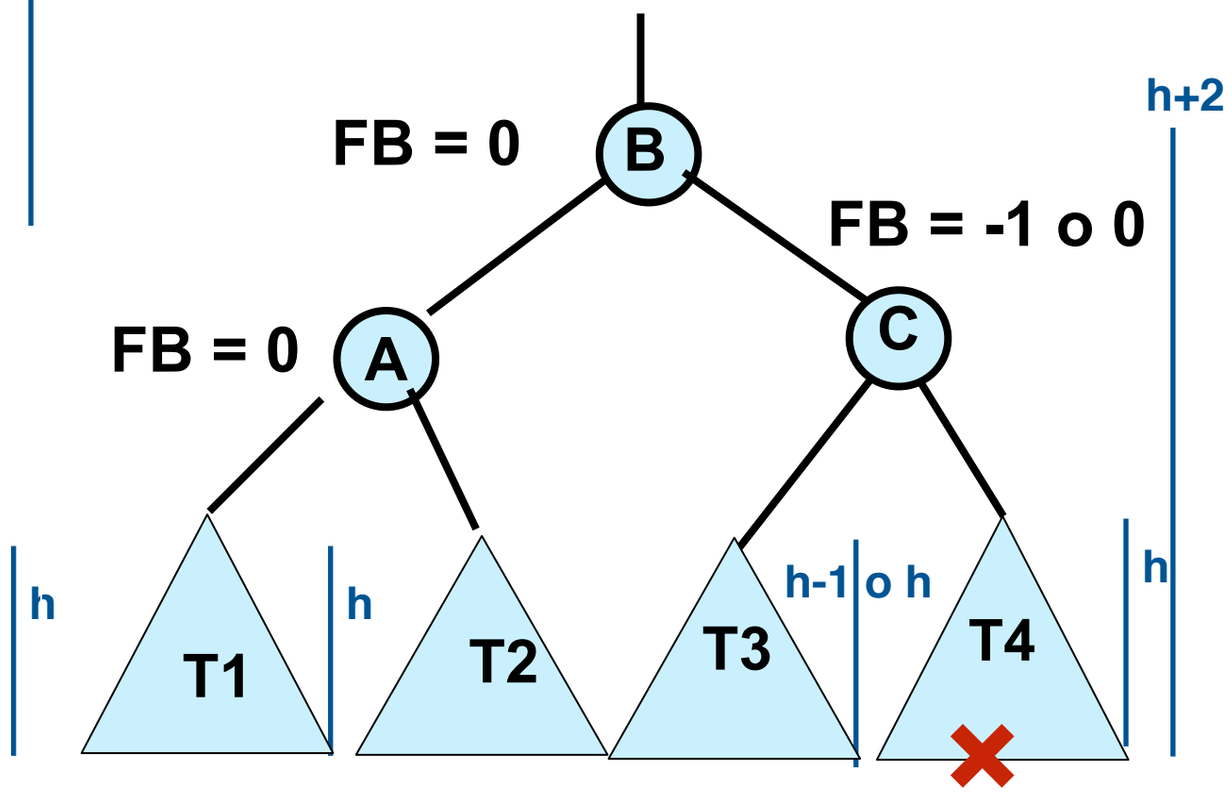
prima rotazione: a sinistra su A



CASO LEFT-RIGHT

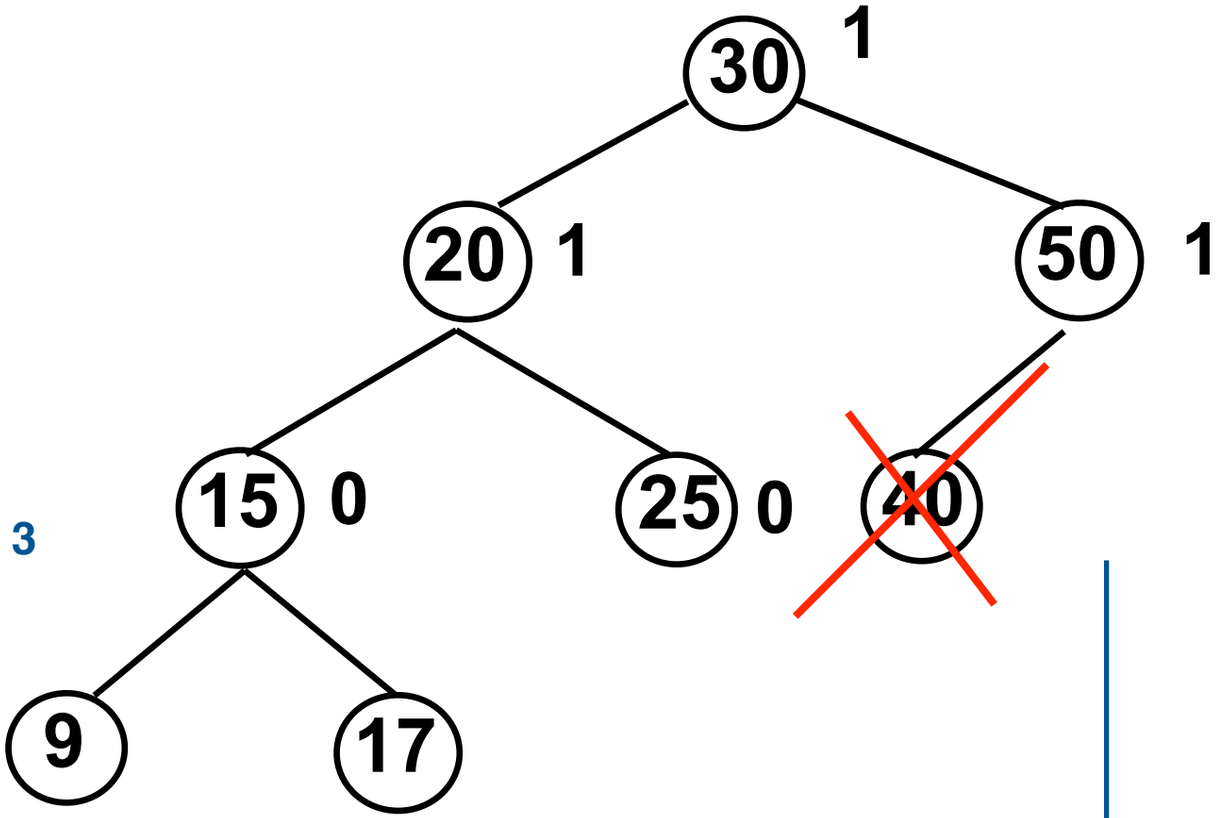


seconda rotazione: a destra su C

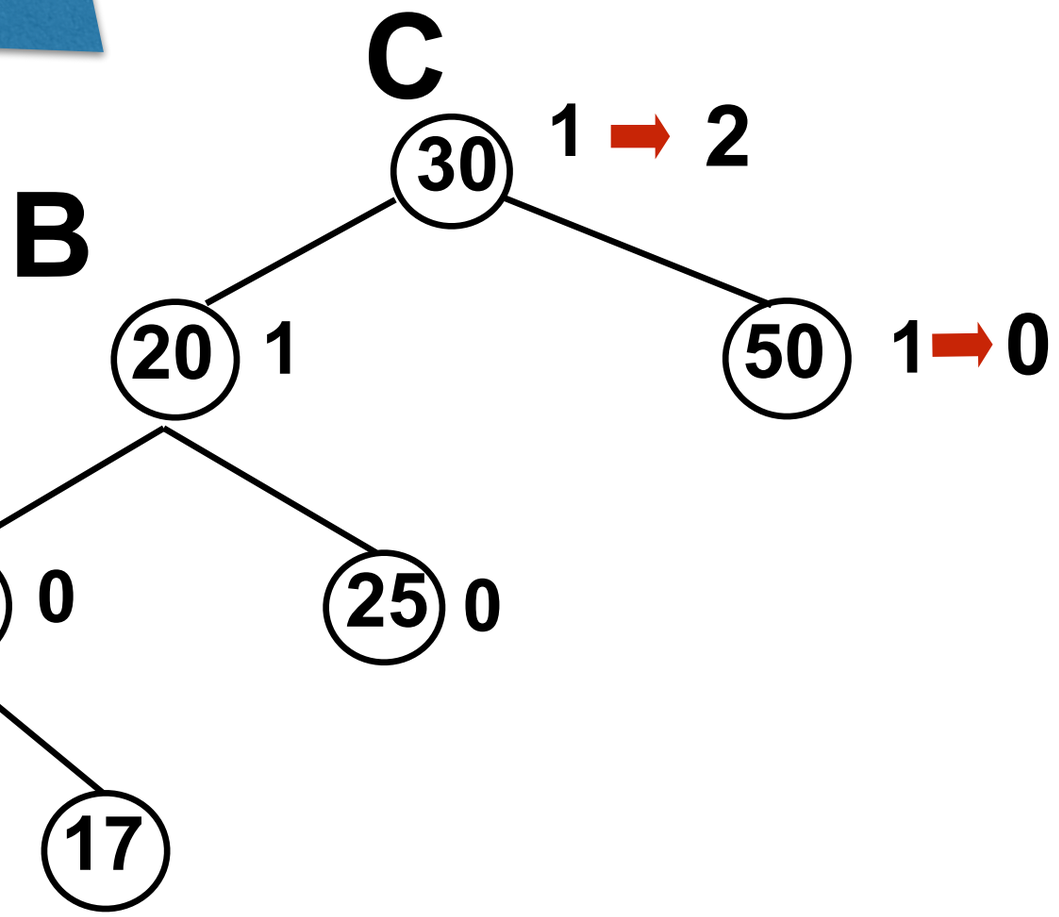


Dopo la seconda rotazione l'altezza di B è minore di quella del nodo C prima dell'inserimento, quindi è necessario risalire ancora verso la radice per aggiornare i fattori di bilanciamento ed eventualmente, con ulteriori rotazioni, ribilanciare l'albero!

Esempio cancellazione con sbilanciamento

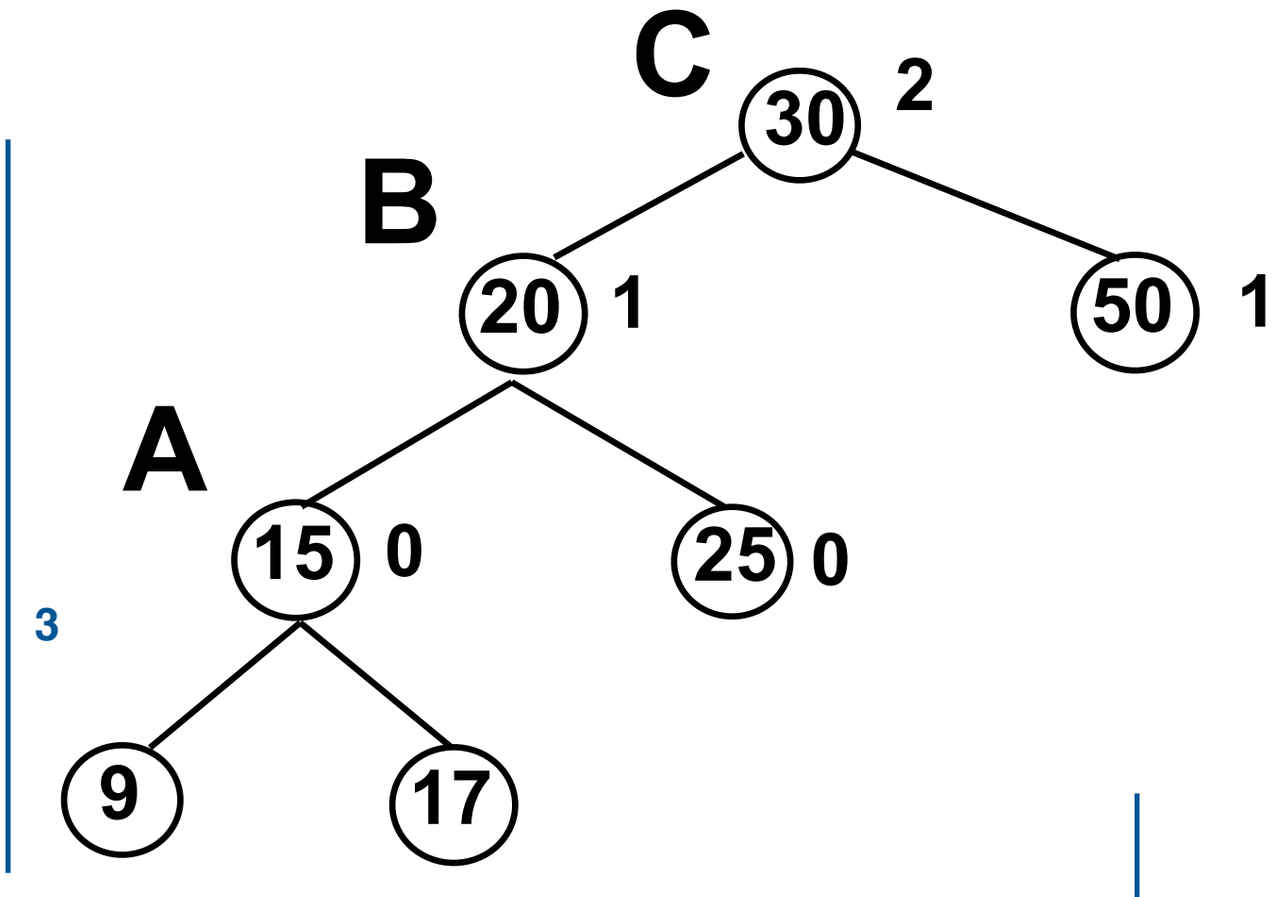


Cancellazione di 40

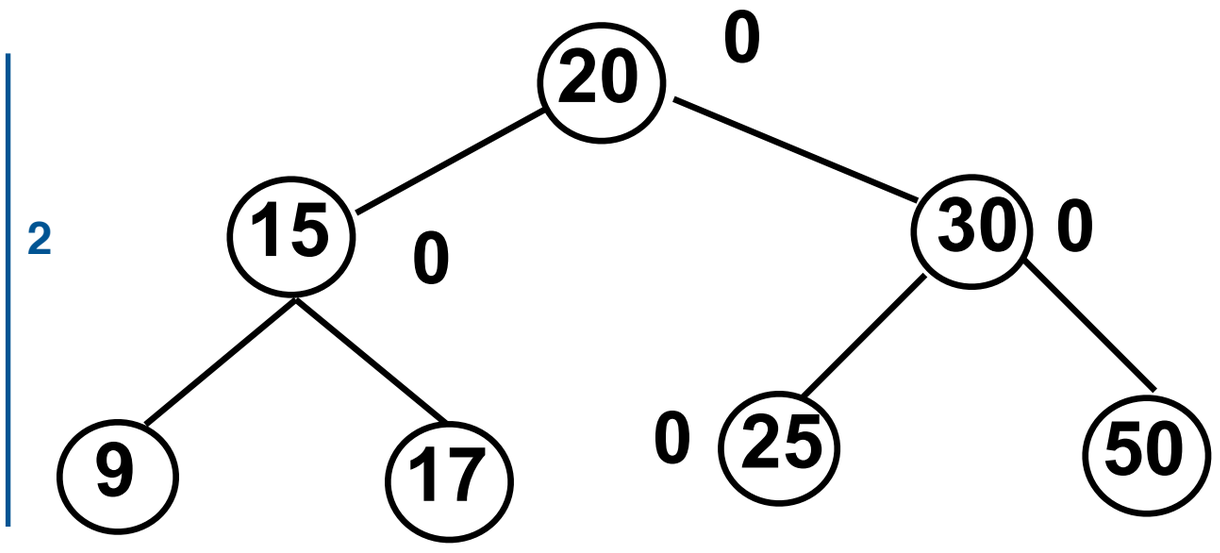


Risalendo da 50, passiamo a un nodo con FB = 2, il cui figlio sinistro ha FB = 1, questo identifica il caso left left2.

Il ribilanciamento

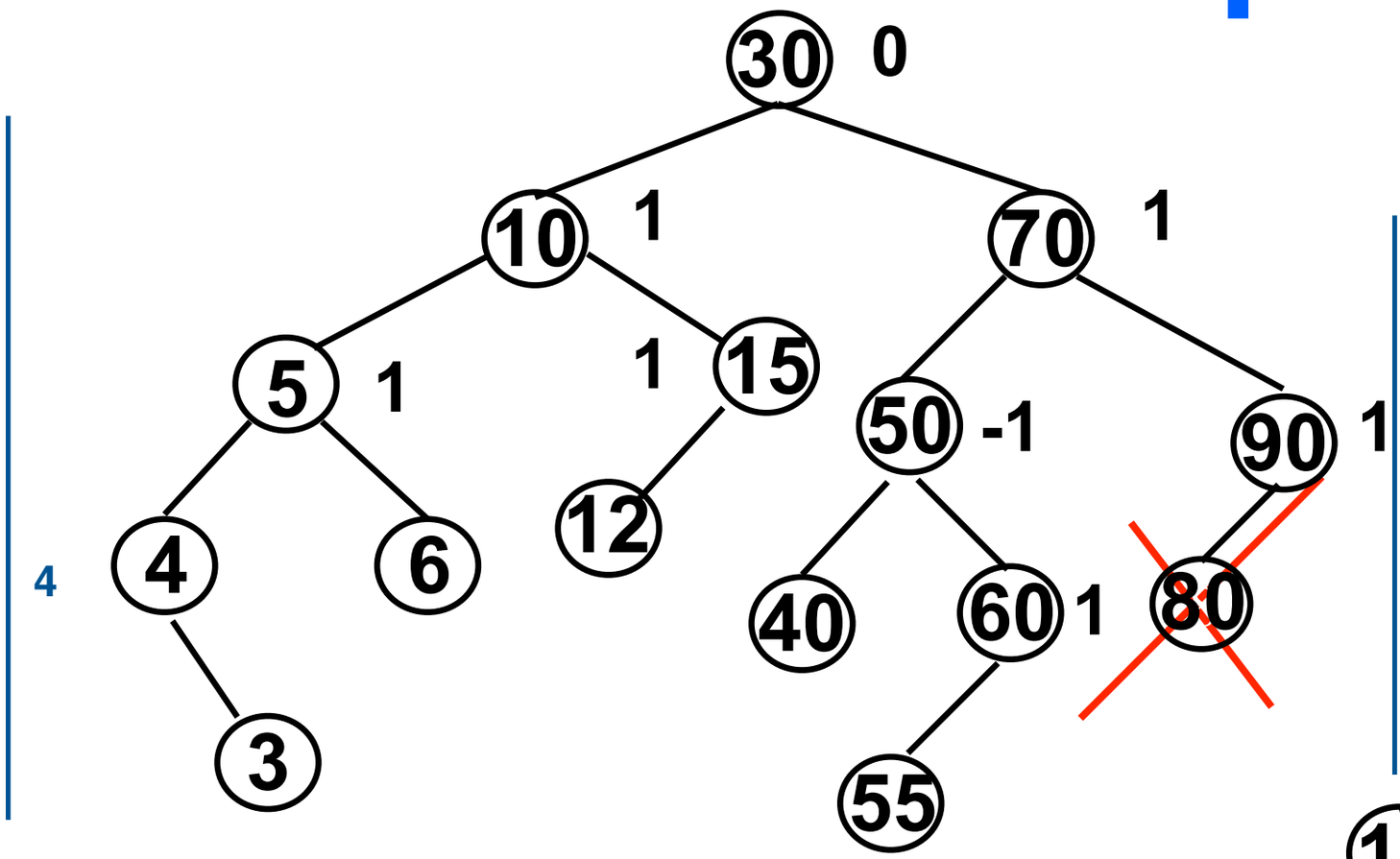


Rotazione a destra su A

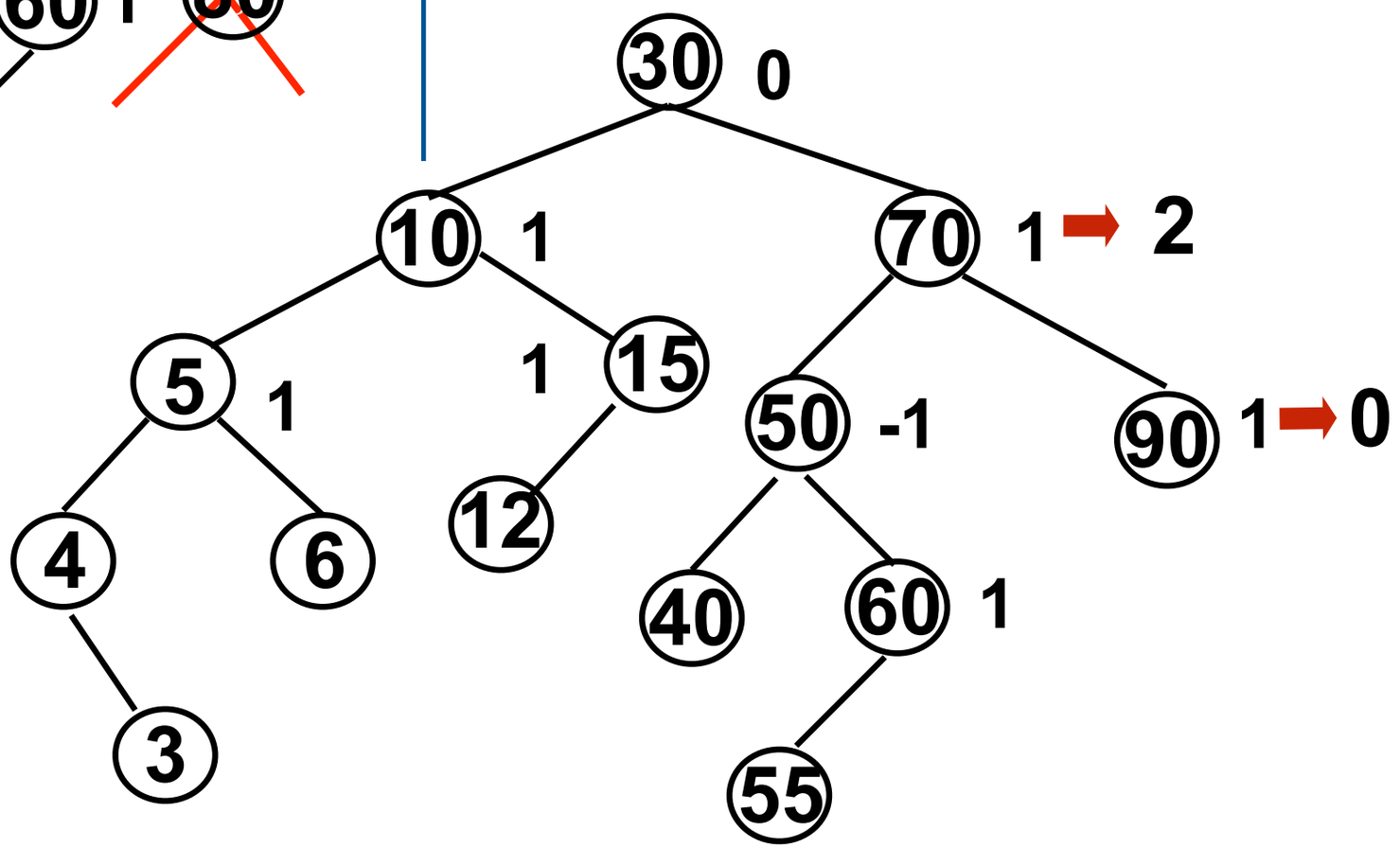


L’altezza dell’albero radicato in B è diminuita di 1 rispetto a quella dell’albero radicato in C che rimpiazziamo. Quindi se C non fosse stata la radice, avremmo dovuto proseguire nella modifica dei FB ed eventualmente eseguire altre rotazioni.

Esempio

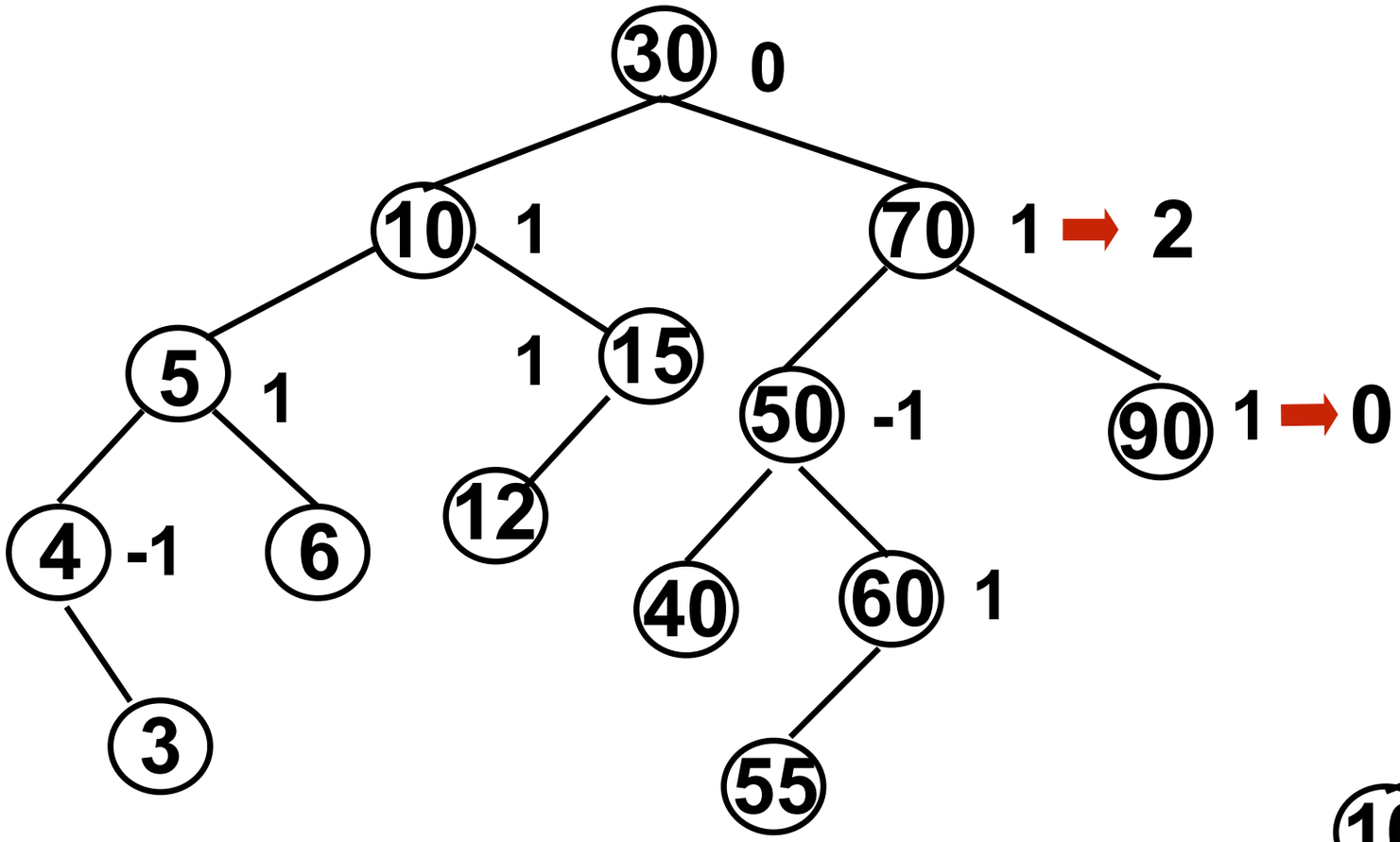


Cancellazione di 80

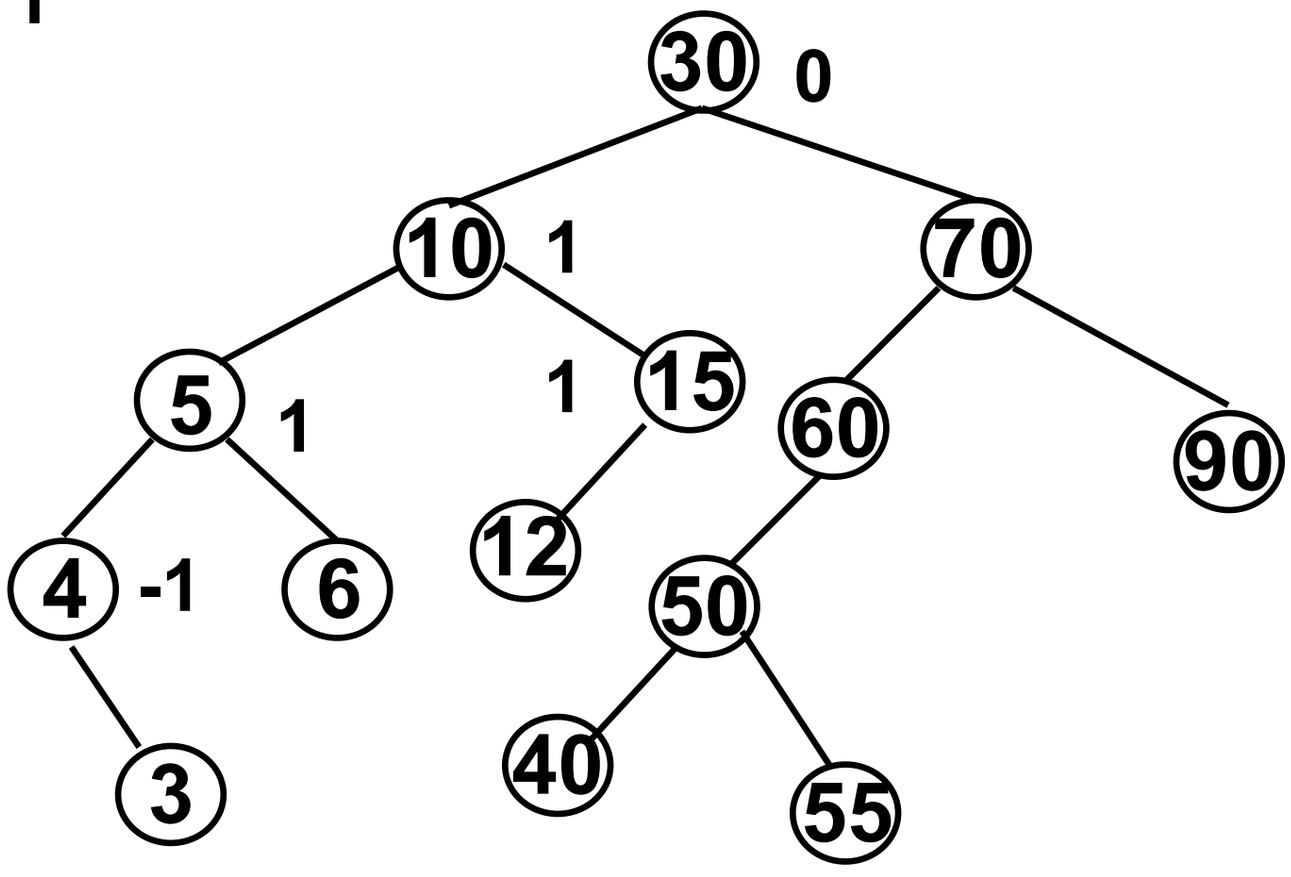


Serve una doppia rotazione:
il nodo con FB = 2 ha il figlio sinistro con FB = -1

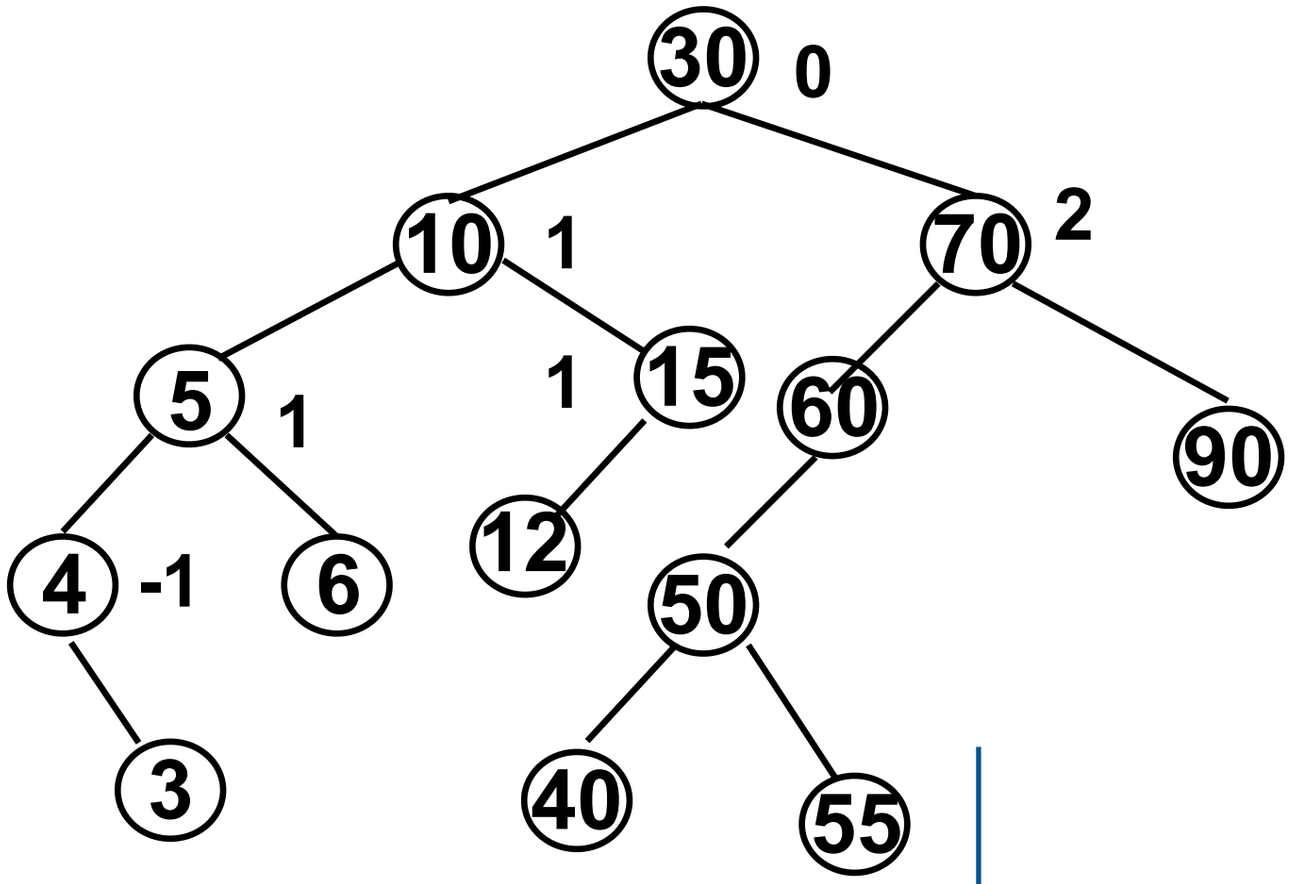
Esempio



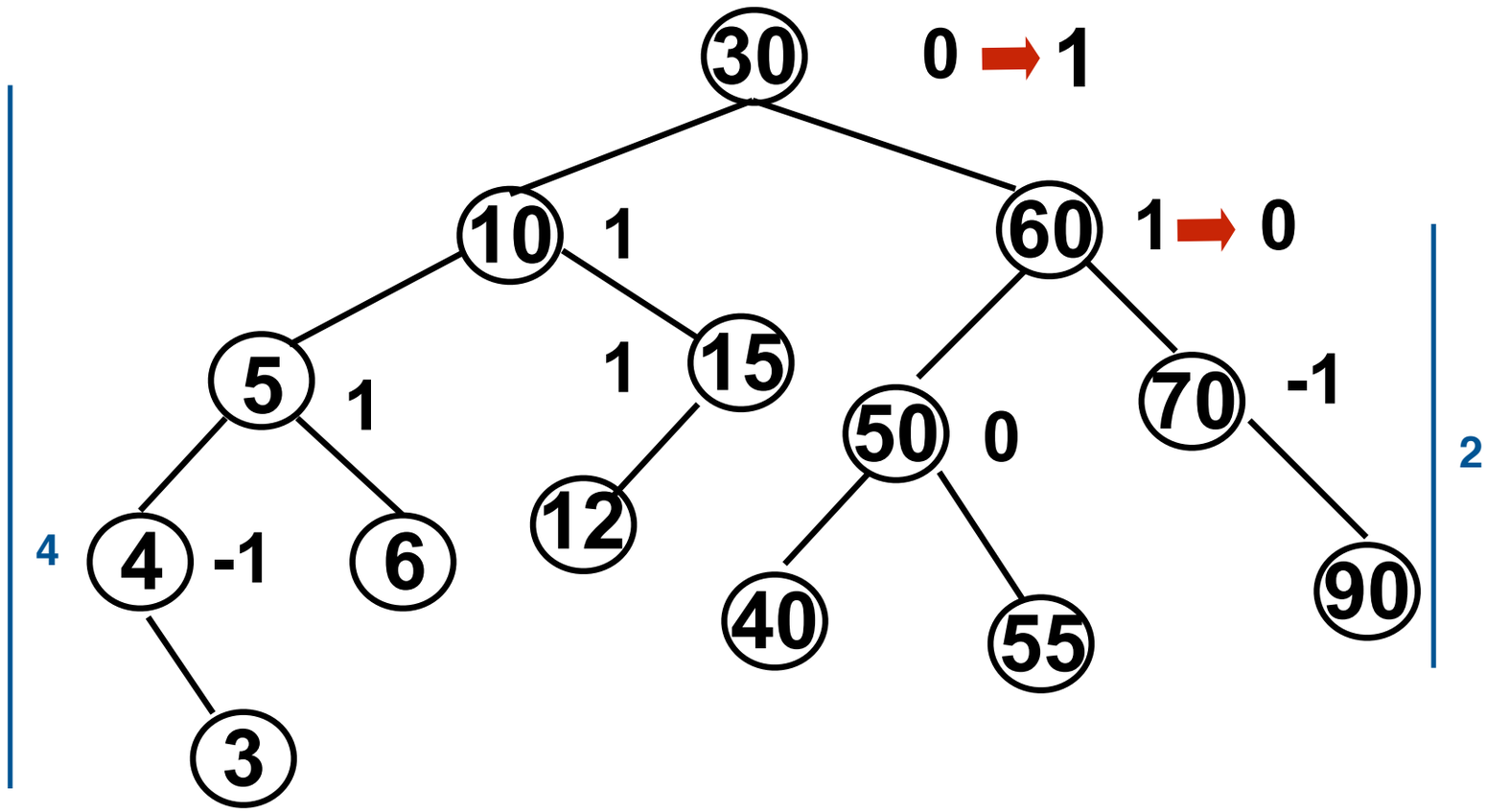
Rotazione a sinistra su 50



Esempio



Rotazione a destra su 70



Pseudocodice ribilanciamento

Ribilanciamento(T)
input: T è il riferimento al primo antenato del nuovo nodo inserito il cui fattore di bilanciamento è illegale, cioè -2 o 2
prec: T è un AVL, salvo la violazione in T
post: l'albero radicato in T è ribilanciato

```
if (T.bf == 2)   %L'altezza è aumentata a sinistra
     P=T.left
     if (P.bf == -1) %L'altezza è aumentata a destra: siamo nel caso Left Right
          then  leftRotate(P) % ci si riconduce al caso Left Left
     rightRotate(T); %  il bf è 0 o 1, l'altezza è aumentata a sinistra, caso Left Left
                        (è 1 nel caso inserimenti, può essere 0 nel caso di cancellazioni)

if (T.bf == - 2) %L'altezza è aumentata a destra
      P=T.right
     if (P.fb) == 1  %L'altezza è aumentata a sinistra: siamo nel caso Right Left
          then  rightRotate(P) % ci si riconduce al caso Right Right
     leftRotate(T);  % il bf è 0 o -1 l'altezza è aumentata a destra, caso Right Right
                       (è 1 nel caso di inserimenti, può essere 0 nel caso di cancellazioni)
```

Conclusione

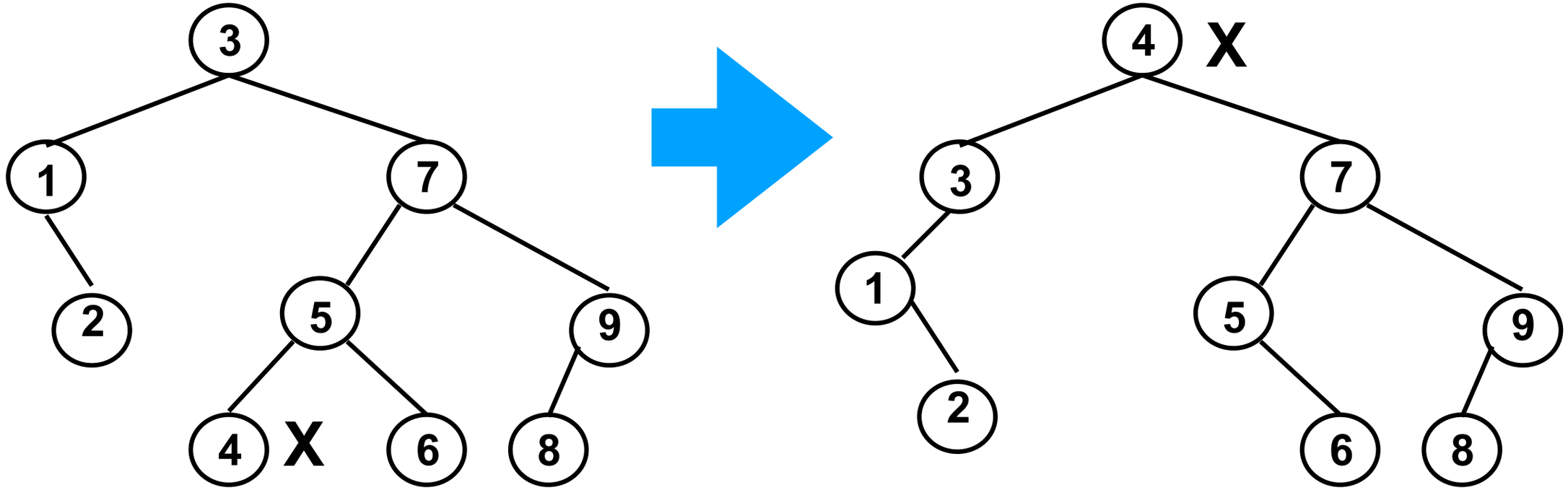
La cancellazione prevede 6 casi: 4 casi di singola rotazione e 2 di doppia: left-left1, left-left2, left-right, right-right1, right-right2, right-left

La funzione di cancellazione viene modificata in modo tale che risalendo verso la radice si aggiornano i fattori di bilanciamento nei nodi.
Dopo ogni aggiornamento si controlla se il valore del FB (la differenza tra le altezze) è legale , se non lo è si chiama la funzione di ribilanciamento.

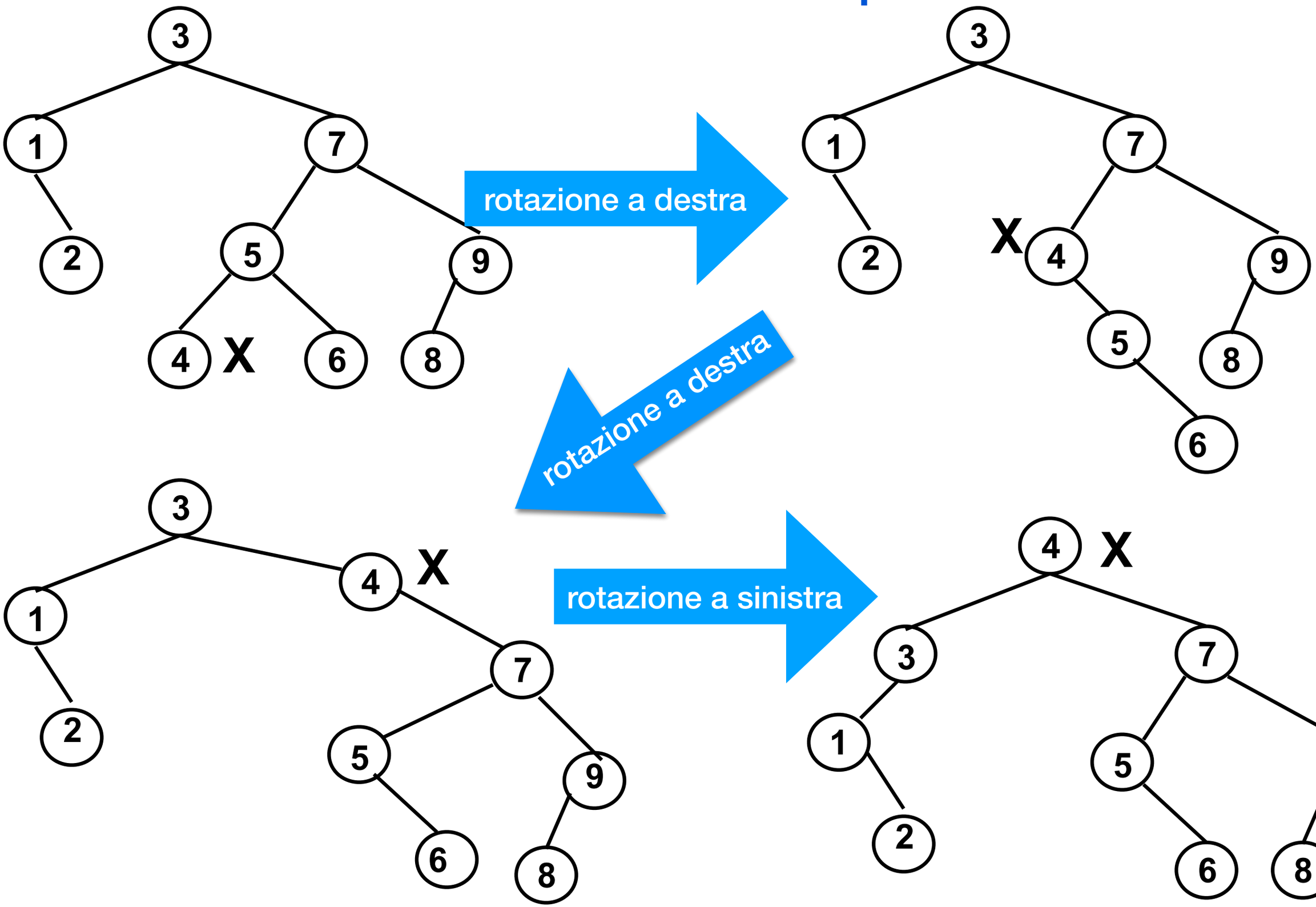
ABR con nuova radice

Preso un ABR T (con chiavi tutte distinte) e un nodo X, vogliamo trasformare T in un nuovo ABR che ha le stesse chiavi di T ma il nodo X alla radice.

Sugg. Si utilizzino le rotazioni.

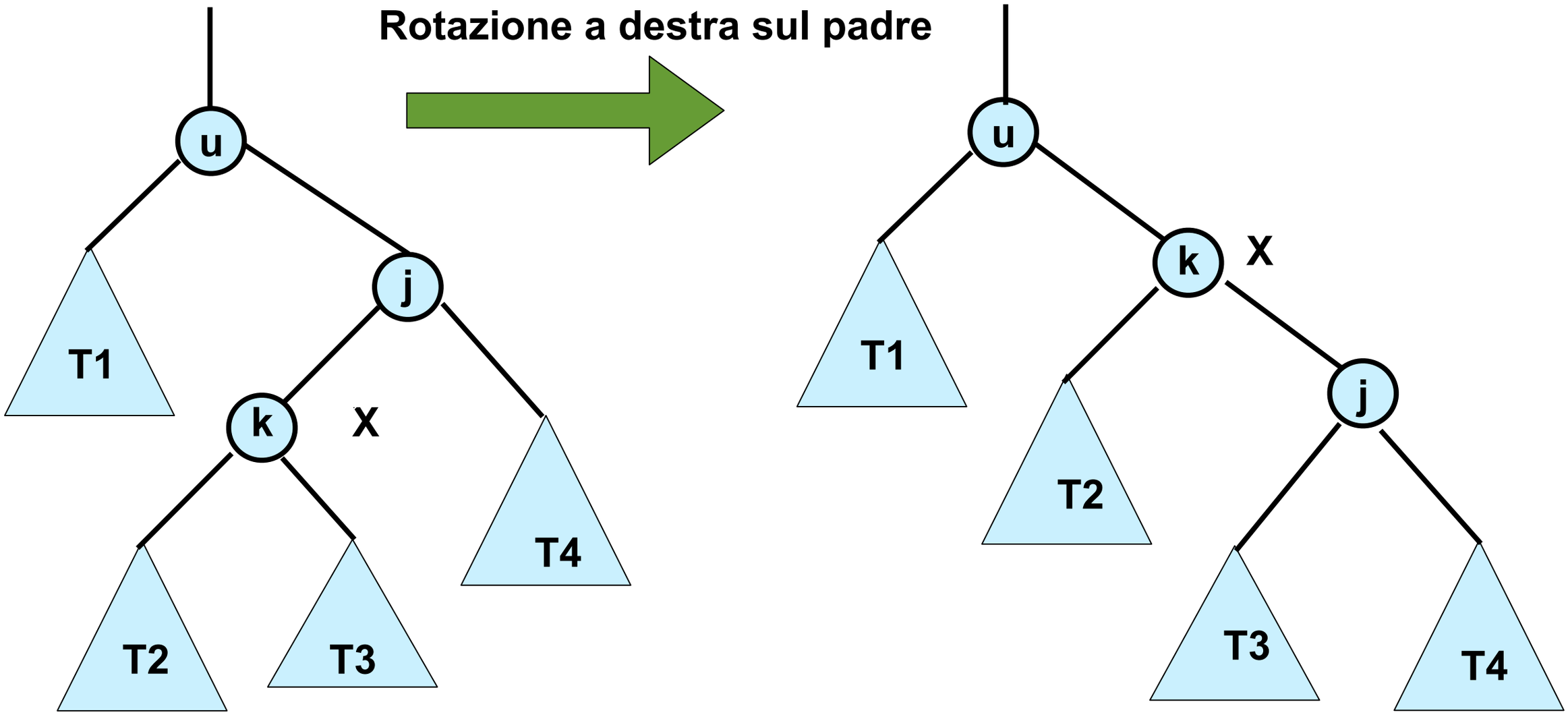


ABR con nuova radice: esempio continutato



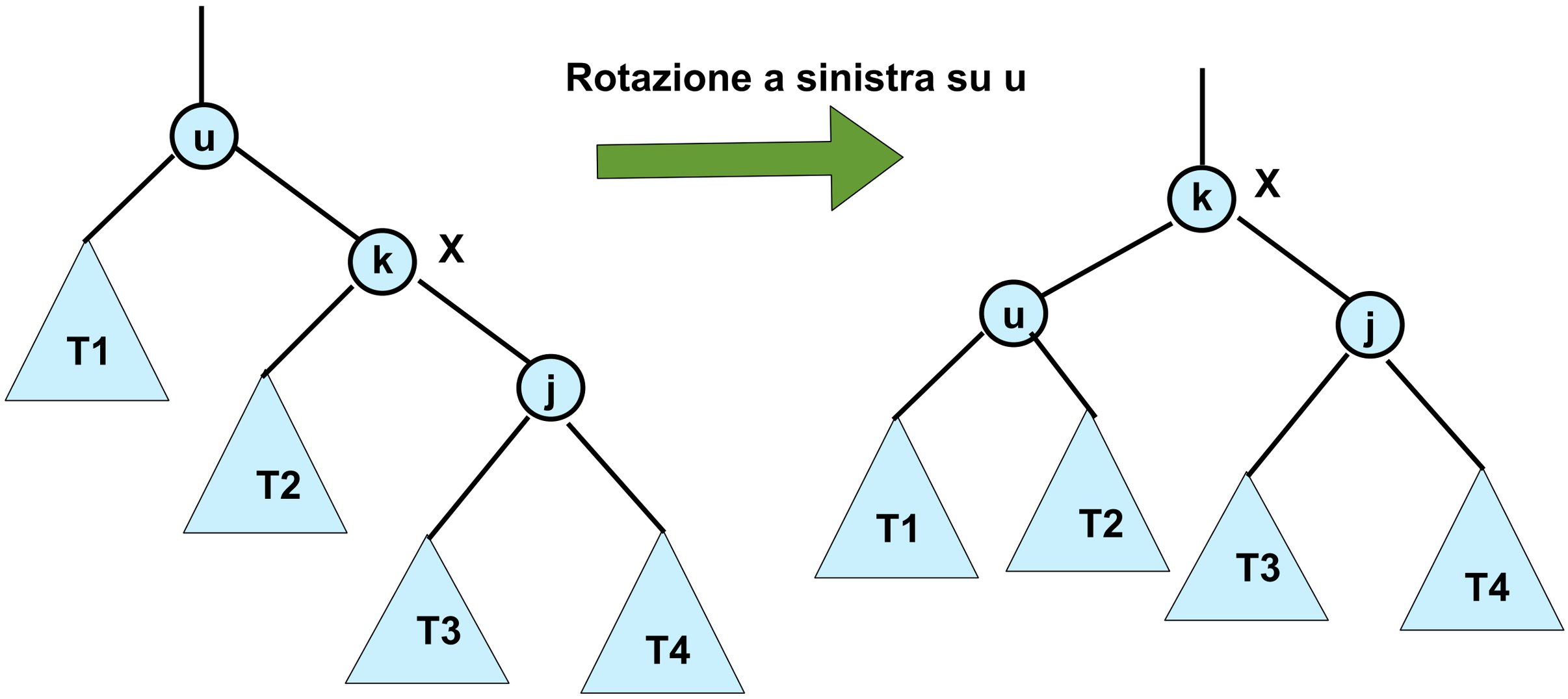
X alla radice, con X figlio sinistro

Se X è figlio sinistro, facciamo una rotazione a destra, altrimenti a sinistra.
Ogni rotazione fa risalire X di un livello verso la radice.



u può essere figlio sinistro o destro

X alla radice, con X figlio destro



Ogni rotazione diminuisce di uno la distanza di x dalla radice e costa Θ(1), quindi il tempo di esecuzione è Θ(h) nel caso peggiore, O(h) in tutti i casi.

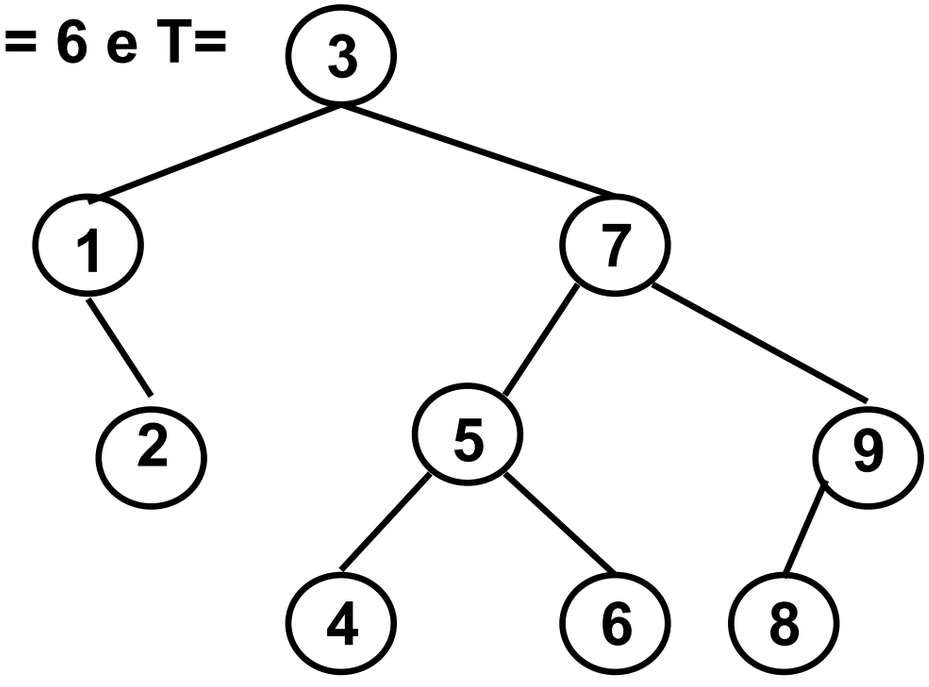
Pseudocodice

```
NuovaRadice(T,X)
input: T è il puntatore alla radice di un albero binario con chiavi intere e X è un nodo di T
prec: T è un ABR
output: il puntatore alla radice di un albero binario con le stesse chiavi di T, ma con il nodo X alla radice.
Y = X.p
   while Y ≠ nil
      if Y.left ≠ NIL and T.left==X then fai una rotazione a destra su Y
      else  fai una rotazione a sinistra su Y
      Y = X.p
return X
```

Split ABR

Preso un ABR T (con chiavi tutte distinte) e una sua chiave k, lo split di T intorno a k produce due ABR T1 e T2 il primo contenente gli elementi più piccoli di k e l'altro i più grandi.
Come realizzeresti uno split e quanto costa in termini di tempo la soluzione proposta?

Esempio con k = 6 e T=



Split ABR

Preso un ABR T (con chiavi tutte distinte) e una sua chiave k, lo split di T intorno a k produce due ABR T1 e T2 il primo contenente gli elementi più piccoli di k e l'altro i più grandi.
Come realizzeresti uno split e quanto costa in termini di tempo la soluzione proposta?

Sol. Si porta k alla radice, utilizzando le rotazioni, poi basta prendere i due sotto alberi della radice come nuovi alberi.

Tempo di esecuzione O(h)

Split ABR: pseudo codice

Split(T,k)
input: T è il puntatore alla radice di un albero binario con chiavi intere, k un intero
prec: T è un ABR e k è una chiave di T
output: il puntatore alle radici di due ABR T1 e T2, con T1 che contiene le chiavi di T minori di k e T2 le maggiori.

```
p = Cerca(T,k)
NuovaRadice(T,p)
T1 = p.left
T2 = p.right
return (T1,T2)
```

NuovaRadice(T,X)
input: T è il puntatore alla radice di un albero binario con chiavi intere e X è un nodo di T
prec: T è un ABR
output: il puntatore alla radice di un albero binario con le stesse chiavi di T, ma con il nodo X alla radice.

Cerca(T,k)
input: T è il puntatore alla radice di un albero binario con chiavi intere e k è un intero
prec: T è un ABR
output: il puntatore al nodo di chiave k in T se presente, NIL altrimenti.

Analisi algoritmi ricorsivi

Posto j-i+1 = n

Esercizio1(A,i,j)
input: A è un vettore di interi e i e j sono interi non negativi.
1. k ← i
2. while (k ≤ j) do
3. h ← i
4. while (h ≤ j) do
5. A[h] ← A[h] +1
6. h ← h +1
7. k ← k +1

Esercizio2 (A,i, j)
if (i<j)
then numElem ← j - i+1
Esercizio2(A, i, i + numElem/4)
Esercizio2(A, j - numElem/4,j)
Esercizio1(A, i, j)

Analisi algoritmi ricorsivi

```
Esercizio1(A,i,j)
 prec: A è un vettore di n elementi e i
e j sono indici compresi tra 1 e n.
1. k = i
2. while (k ≤ j) do
3.          h = i
4.      while (h ≤ j) do
5.                  A[h] = A[h] +1
6.                  h =h +1
7.          k = k +1
```

Posto j-i+1 = n

eseguito n volte, quindi in $\Theta(n^2)$

eseguito n volte

```
Esercizio2 (A,i, j)
if ( i < j) then n = j - i+1
Esercizio2( A, i, i + n/4)
Esercizio2( A, j - n/4,j )
Esercizio1( A, i, j )
```

tempo di esecuzione Esercizio1 in $\Theta(n^2)$

tempo di esecuzione Esercizio2:

$T(n) = 2T(n/4) + \Theta(n^2)$

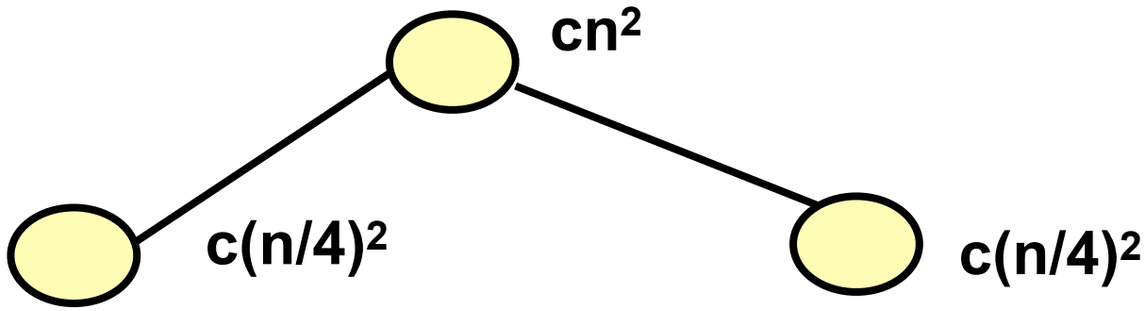
relazione da risolvere

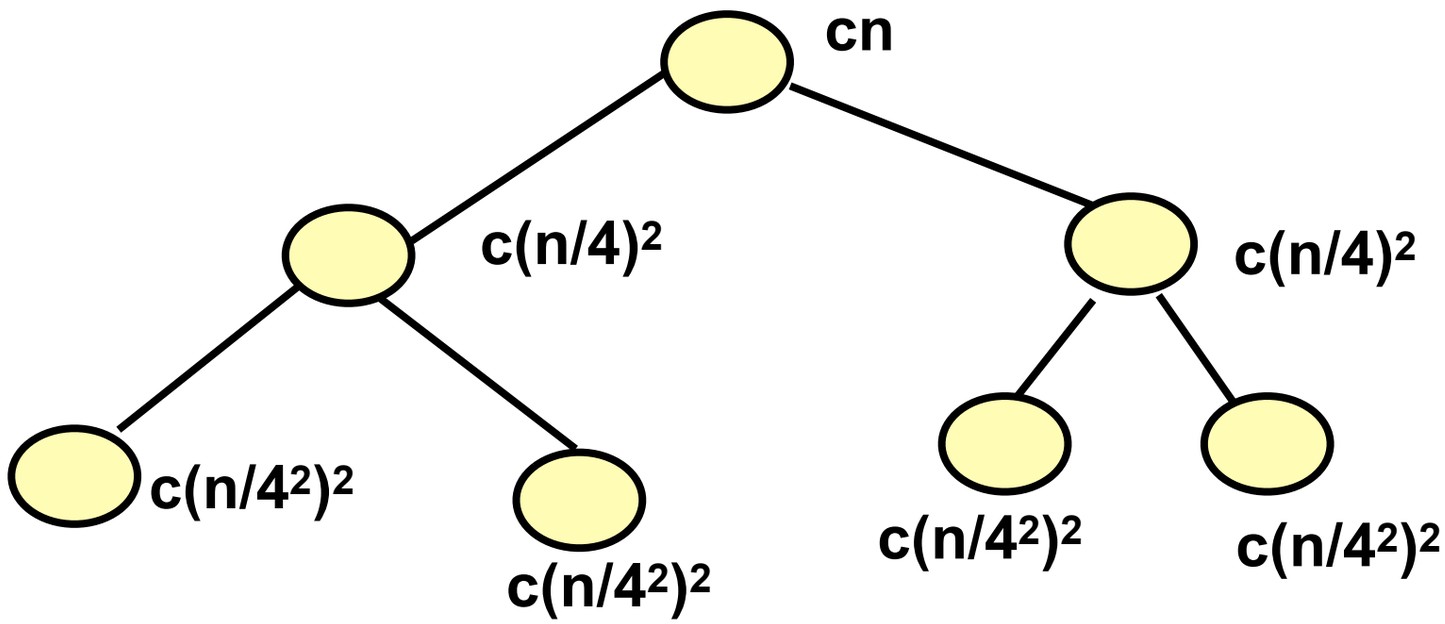
$T(n) = d$ se $n \leq 1$

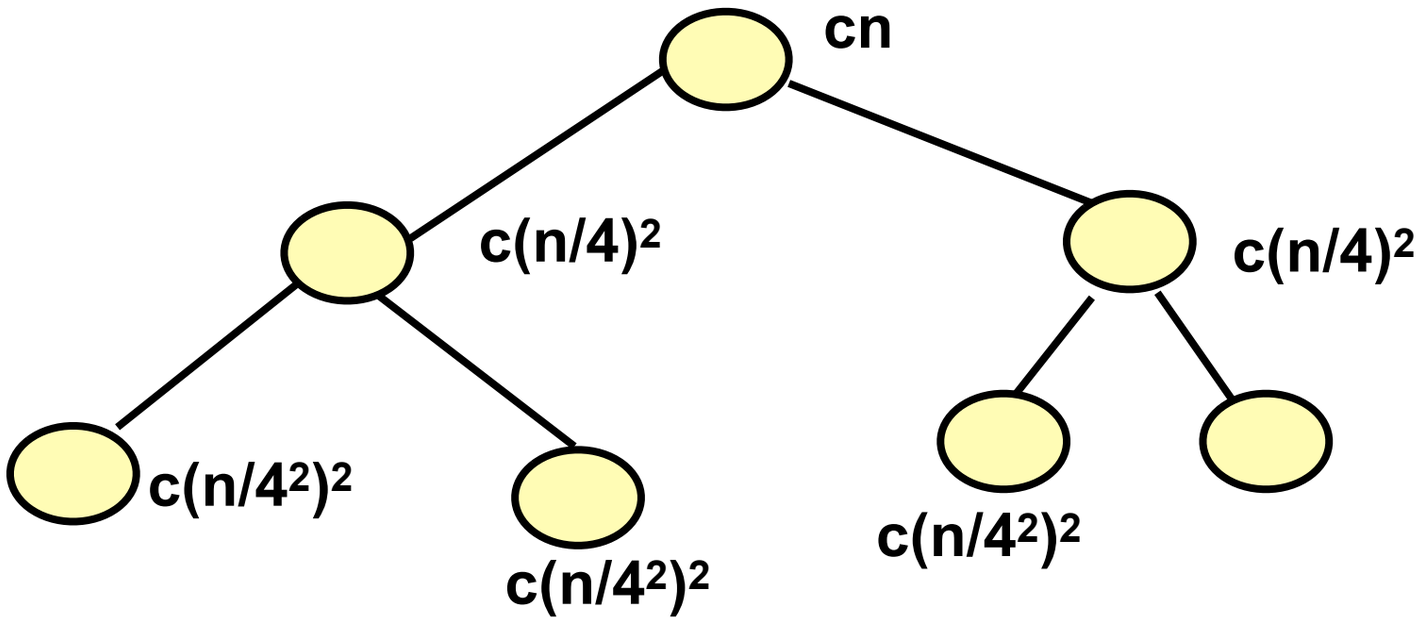
$T(n) = 2T(n/4) + cn^2$ altrimenti

Assumiamo che i + x con $0<x<1$ sia i e anche j-x = j

$T(n) = 2T(n/4) + cn^2$







Ponendo per semplicità $n=4^h=2^{2h}$ in un generico livello i, i nodi saranno etichettati con $c(n/4^i)^2 = cn^2/4^{2i}$ il penultimo livello conterrà nodi etichettati $c(n/4^{h-1})^2 = cn^2/4^{2(h-1)}$ mentre l'ultimo livello ha 2^h nodi etichettati d.

Possiamo dire che $T(n) = 2^h d + \sum_{i=0,\ldots,h-1} c2^i n^2/4^{2i} = 2^h d + \sum_{i=0,\ldots,h-1} c2^i n^2/2^{4i}$

$= n^{1/2} d + cn^2 \sum_{i=0,\ldots,h-1} 1/2^{3i} \leq n^{1/2} d + cn^2 \sum_{i=0,\ldots,\infty} 1/2^{3i} = O(n^2)$, visto che la serie è convergente a una costante.

$T(n) = d$ se $n \leq 1$
$T(n) = 2T(n/4) + cn^2$ altrimenti

Dobbiamo dimostrare per induzione che esistono due costanti positive k e n_0 tali che $T(n) \leq kn^2$ per ogni $n \geq n_0$.

Supponiamo la tesi vera per tutti i valori $m < n$, quindi $T(n/4) \leq k(n/4)^2$. Considerando T(n) si ha:

$$T(n) = 2T(n/4) + cn^2 \leq$$
$$2k(n/4)^2 + cn^2 =$$
$$kn^2/8 + cn^2$$

Cerchiamo ora i valori di k e n che rendano vera la disuguaglianza: $kn^2/8 + cn^2 \leq kn^2$ per ogni $n \geq n_0$.

$kn^2/8 + cn^2 \leq kn^2 \Leftrightarrow$

$kn^2 + 8cn^2 \leq 8kn^2 \Leftrightarrow$

$8cn^2 \leq 7kn^2 \Leftrightarrow$

$8/7c \leq k$ e $n \geq 1$ Quindi $T(n) \leq 2cn^2$ per ogni $n \geq 1 = n_0$, prendendo $k=2c$. Ma per il caso base dovrebbe essere $T(1) \leq 2c$, mentre $T(1) = d$ e quindi prendiamo $k= 2c+d$ e possiamo concludere che $T(n) = O(n^2)$. Poiché il termine additivo fornisce un limite inferiore in $\Omega(n^2)$, concludiamo che $T(n) = \Theta(n^2)$.

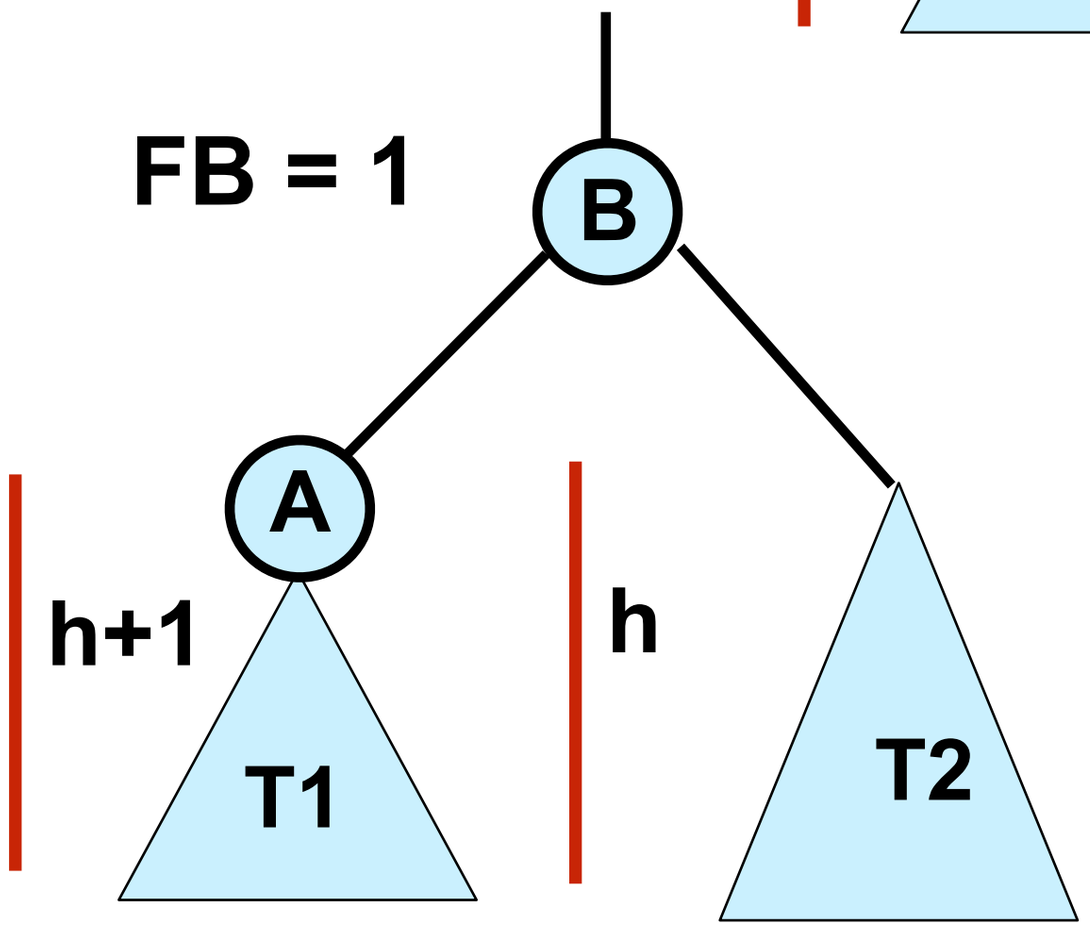
Altezza per AVL

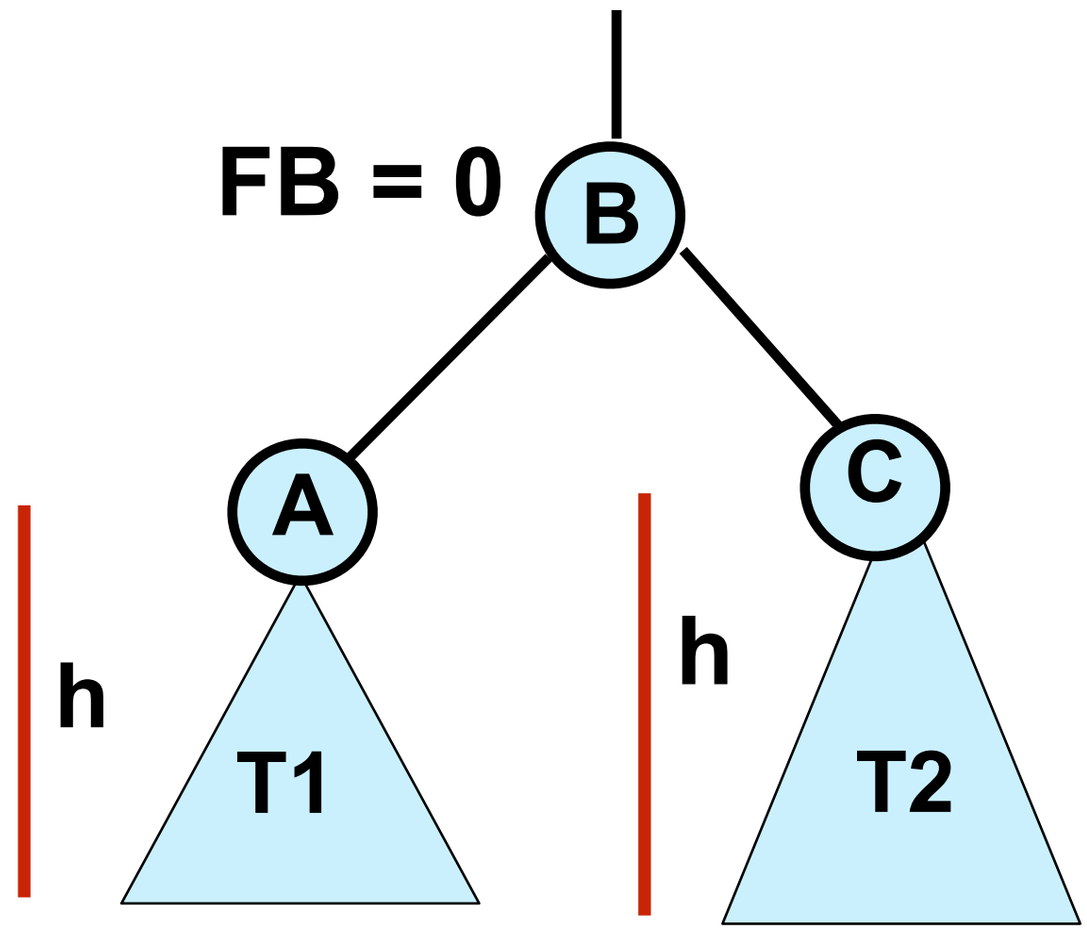
Ogni nodo è implementato con una struttura con tre campi puntatori, al figlio sinistro, al figlio destro e al padre, e due campi a valore intero, la chiave e il fattore di bilanciamento.

E' possibile, in tal caso, calcolare l'altezza di un AVL in O(lg n), invece di O(n).

Soluzione

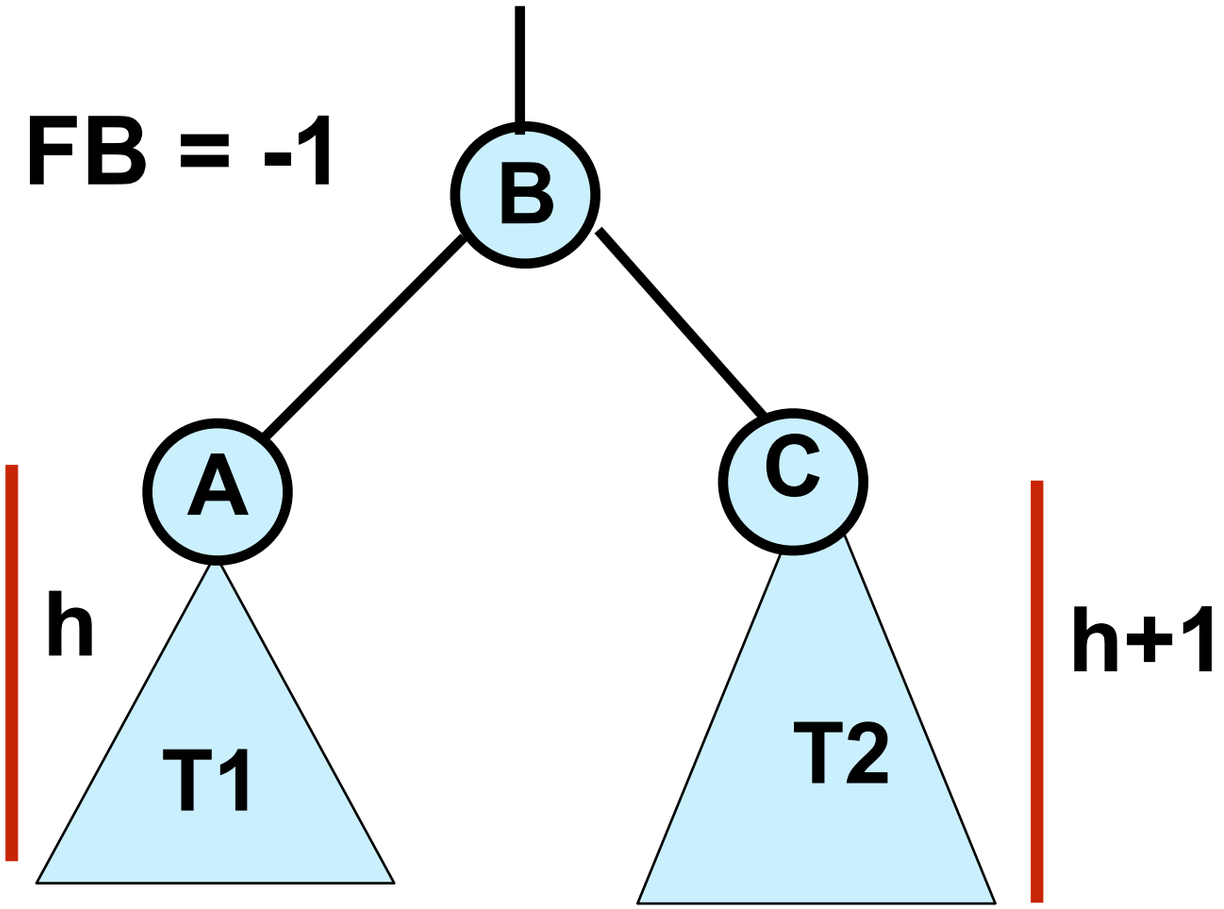
Se il fattore di bilanciamento di un nodo è +1 o 0 si può chiamare la funzione per il calcolo dell'altezza sul sotto albero sinistro, perché di altezza maggiore o uguale del destro.





Soluzione

Se FB = -1 si fa la chiamata sul figlio destro.



Così in ogni passo si scende di un livello dalla radice e si arriverà a una foglia di profondità massima.

Basta quindi incrementare ad ogni passo il valore ottenuto dalla chiamata.

Lo pseudocodice

```
algoritmo AltezzaAVL(T)
input: T è la radice  di un AVL
output: da in output l'altezza di T
  if (T = NULL) return -1
  if (T.FB == 1 or T.FB == 0)
        then return AltezzaAVL(T.left) + 1
  return AltezzaAVL(T.right) + 1
```

Il tempo di esecuzione è in Θ(h), infatti abbiamo una chiamata scendendo dalla radice fino a una foglia di profondità massima.

La relazione di ricorrenza è T(h) = T(h-1) + c, conosciamo e per la quale vale T(n) = Θ(h).

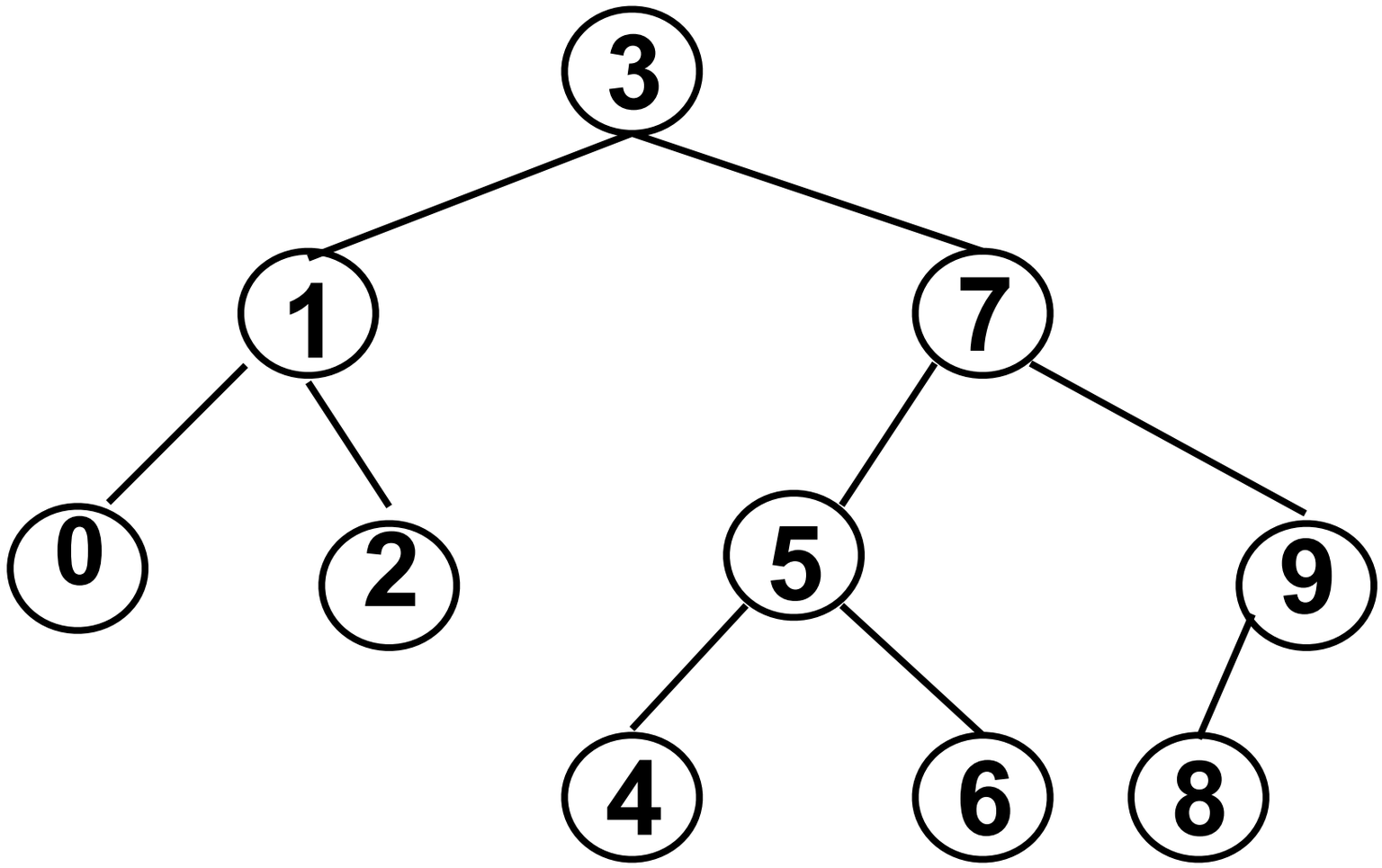
Antenato più vicino

Presi due valori n1 ed n2 ed un ABR T rappresentato tramite puntatori ai figli sinistro e destro, si scriva un algoritmo ricorsivo, che trovi il più vicino antenato dei due nodi aventi chiavi n1 ed n2.

Si può assumere che le due chiavi siano presenti in T. L'algoritmo progettato deve restituire il puntatore all'antenato comune più vicino, cioè quello i cui discendenti non sono antenati comuni dei nodi chiave di n1 e n2.

Si descriva l'idea algoritmica e se ne analizzi la complessità, che dovrebbe essere O(h), dove h denota l'altezza dell'albero.

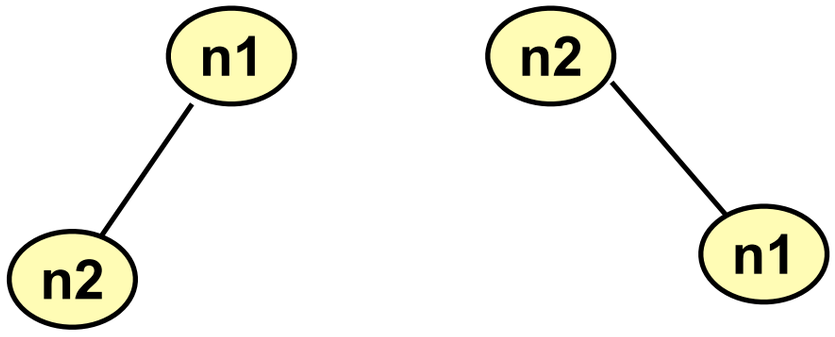
Antenato più vicino esempio



Il più vicino antenato dei due nodi aventi chiavi 5 e 8 è il nodo di chiave 7, mentre per 2 e 9 è il nodo di chiave 3

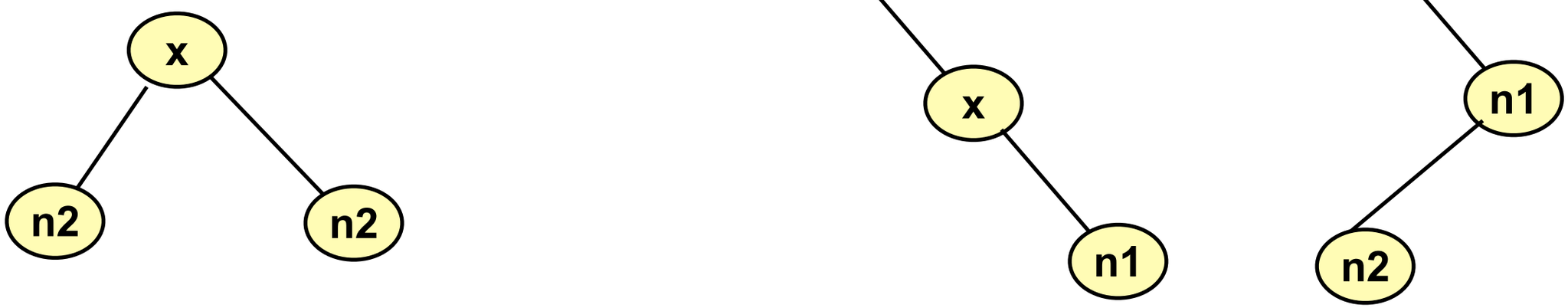
Antenato più vicino

Per l'analisi consideriamo all'inizio alberi piccoli. sappiamo che n1>n2, allora i soli alberi con due nodi sono:



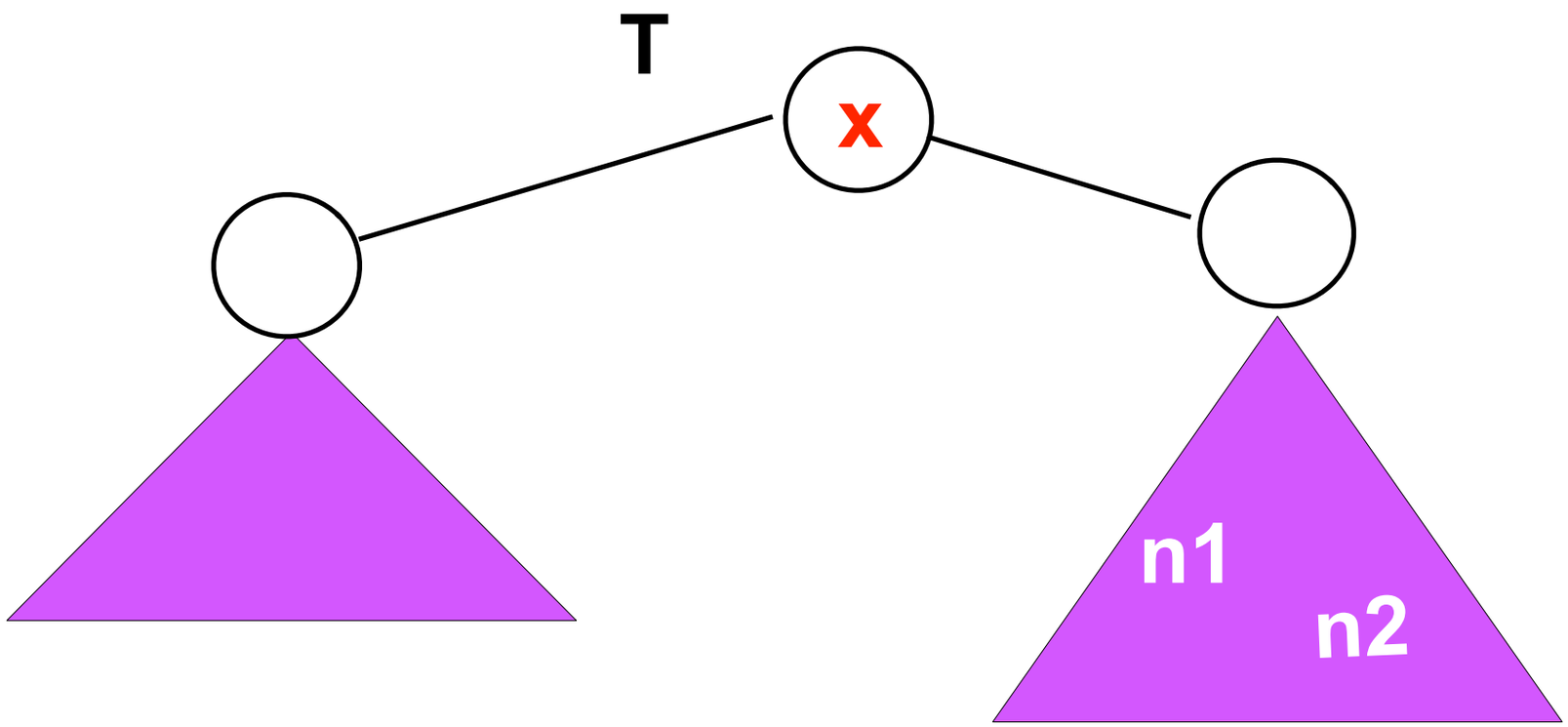
Nel primo il nodo di chiave n1 è il l'antenato comune, nel secondo quello di chiave n2

Con tre nodi si può avere:



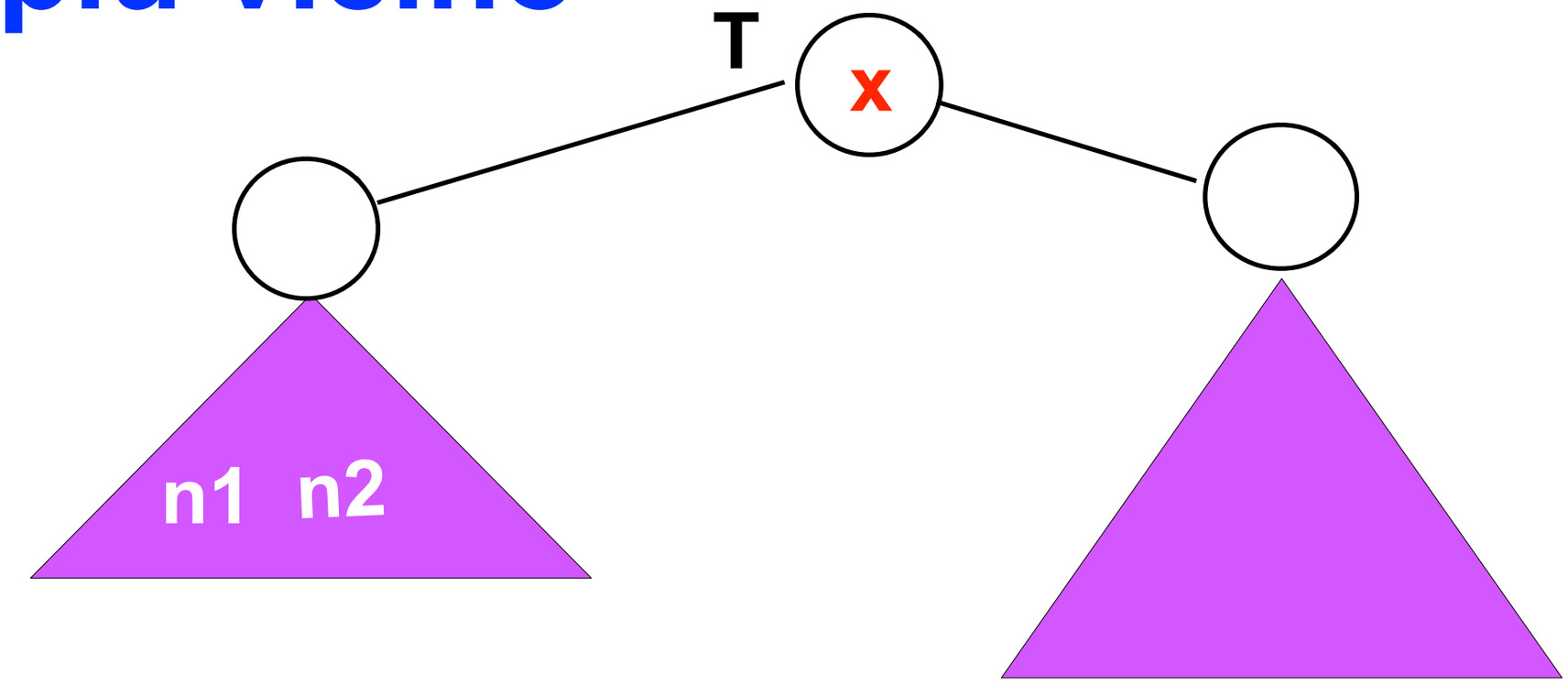
Nel primo albero il nodo di chiave x è l'antenato comune, nel secondo quello di chiave n2, nel terzo quello di chiave n1.

Antenato più vicino



Questo caso è caratterizzato dal fatto che x < n1 e x < n2, e certamente il nodo T di chiave x non è l'antenato comune più vicino, perchè se non altro potrebbe esserlo il figlio destro, si deve quindi a cercarlo nel sotto albero destro

Antenato più vicino

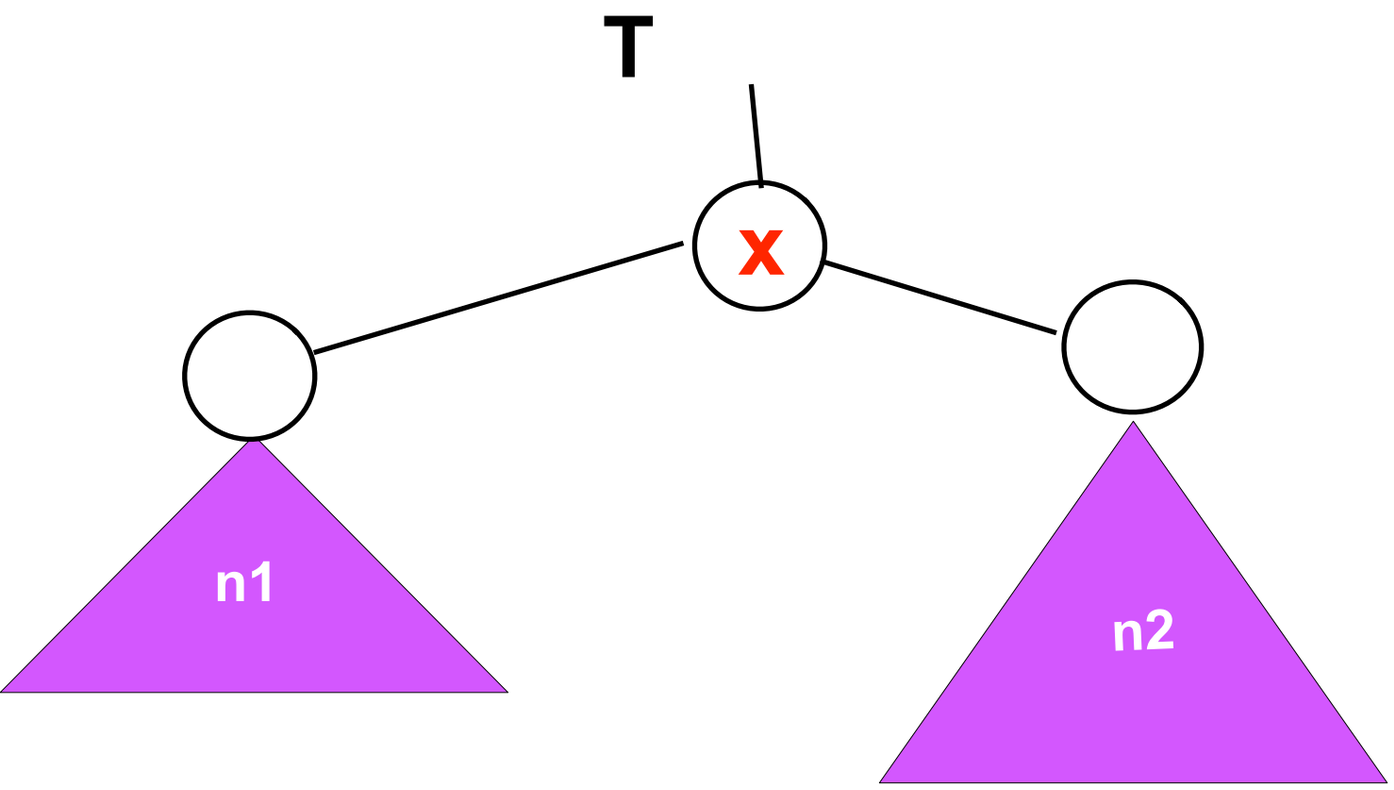


Questo è il caso simmetrico, per il quale valgono considerazioni analoghe e che è caratterizzato dal fatto che x > n1 e x > n2.
Si deve dirigere la ricerca nel sotto albero sinistro.

Infatti se x<n1 e x<n2, certamente il nodo T di chiave x non è l'antenato comune più vicino, perchè se non altro potrebbe esserlo il figlio sinistro, si deve quindi a cercarlo nel sotto albero destro

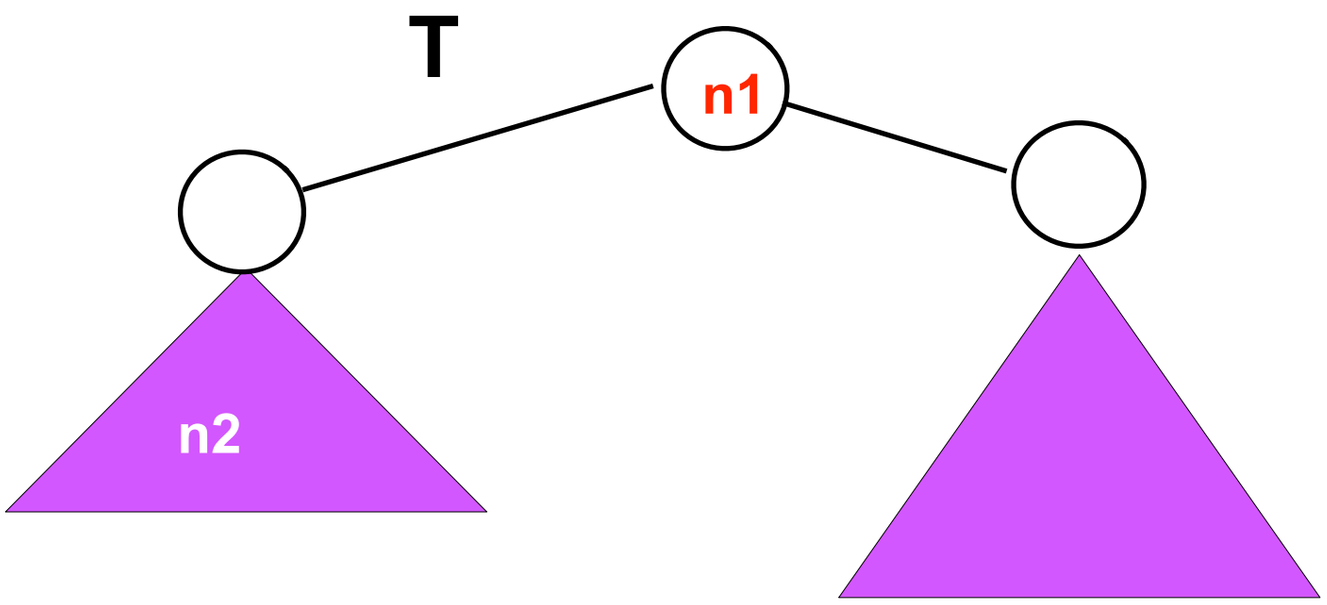
Antenato più vicino

Generalizzando si vede che se x è un nodo il cui sottoalbero sinistro contiene n1 e il cui sottoalbero destro contiene n2, allora x è l'antenato più vicino.

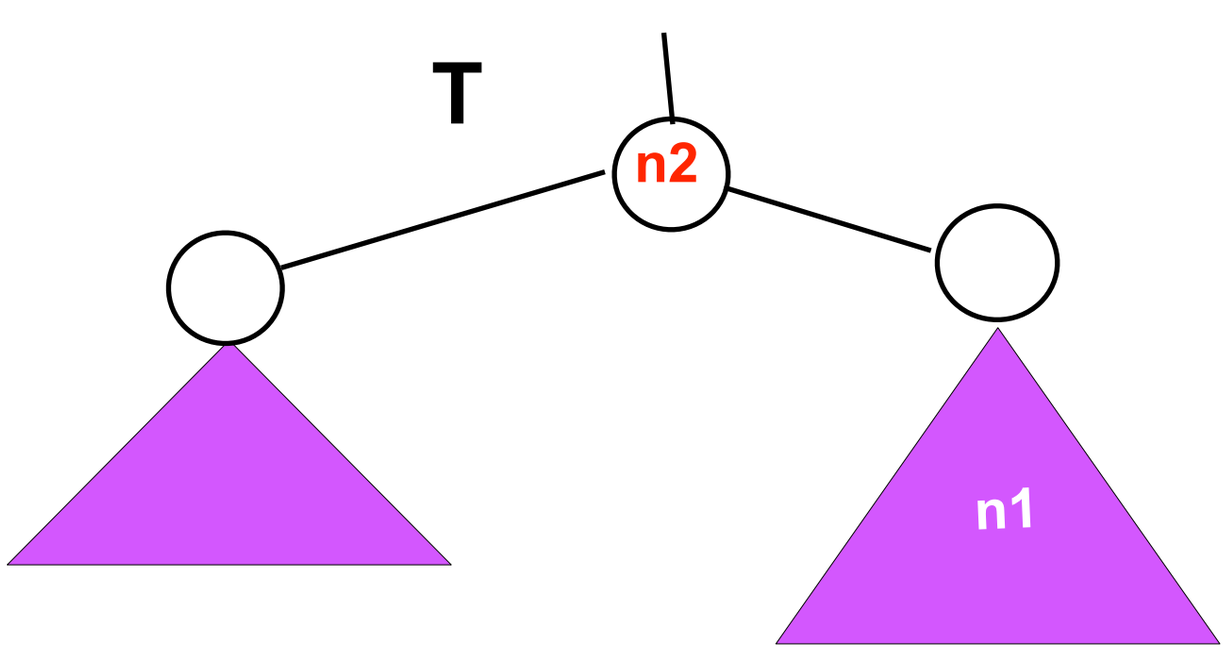


Osserviamo che questo caso è caratterizzato dal fatto che n1<x<n2.
Chiaramente il nodo di chiave x è l'antenato comune più vicino, visto che ogni antenato di n1 nel sottoalbero destro di x non lo è di n2 , e viceversa.
Quindi in questo caso l'algoritmo deve restituire T

Antenato più vicino



Mentre in questo caso la chiave di T è uguale a n1 e maggiore di n2, e T è il nodo da restituire



Il nodo T qui ha la chiave uguale a n2 e maggiore di n1, e T è il nodo da restituire.

Quindi il caso base è
if (n1 ≤ T.key and T.key ≤ n2) then return T

Antenato più vicino: pseudocodice

Antenato(T,n1,n2)

input: T è un albero binario (il riferimento alla radice), n1 e n2 due valori interi

prec: T è un ABR, n1 < n2 sono due chiavi presenti in T

output: dà il puntatore all'antenato comune più vicino (quello i cui discendenti non sono antenati comuni dei nodi chiave di n1 e n2)

```
if (n1 ≤ T.key and T.key ≤ n2) then  return T
if (T.key < n1  then
            return Antenato(T.right,n1,n2)
if (T.key > n2)  then
            return    Antenato(T.left,n1,n2)
```

Correttezza: deriva dalle considerazioni fatte prima.

Antenato più vicino: analisi

Complessità: Il tempo di esecuzione al di fuori delle chiamate è costante, inoltre si fa al più una chiamata su uno dei due figli quindi al più si eseguono, nel caso peggiore, h chiamate se h è l'altezza dell'albero.
Si può concludere che si ha un tempo di esecuzione asintotico in O(h).