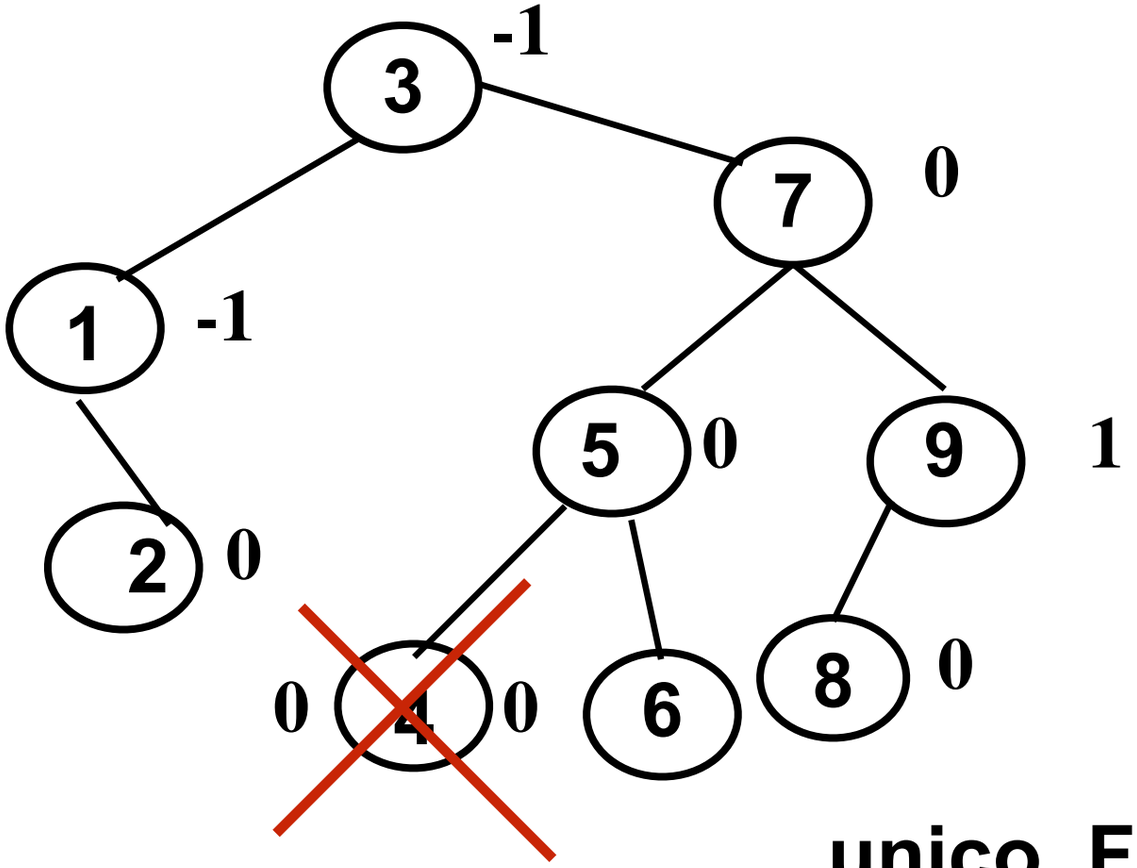
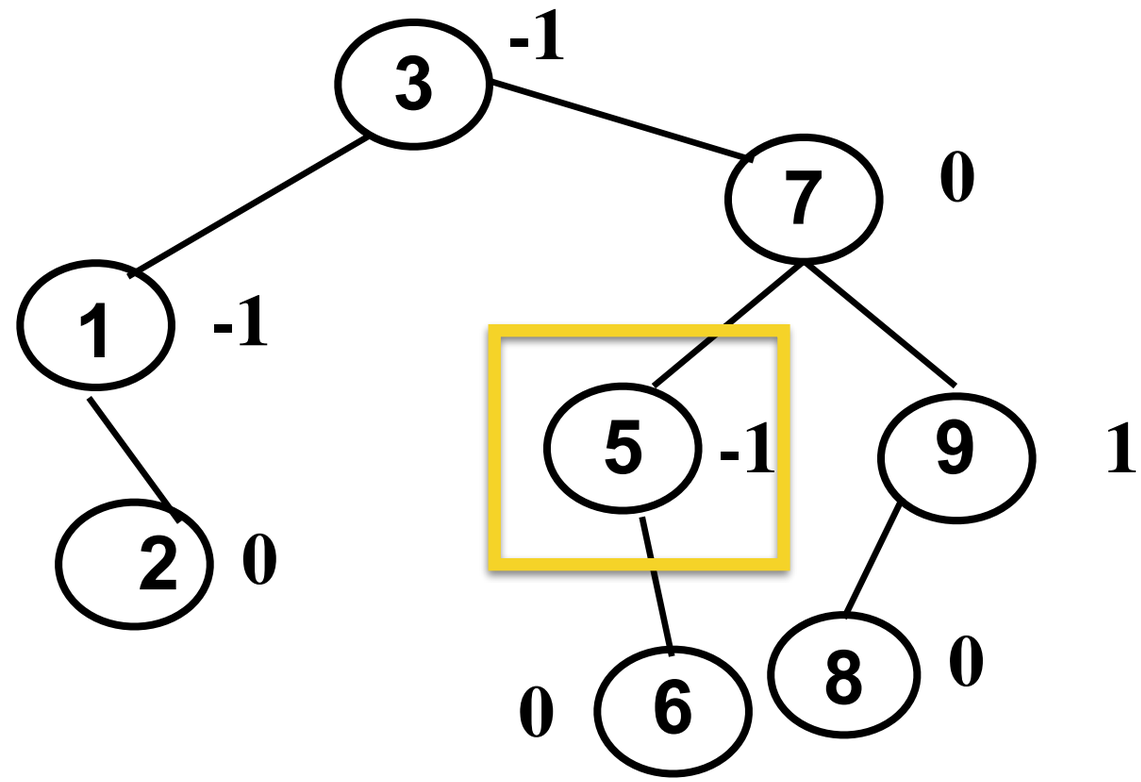
La cancellazione in un AVL

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





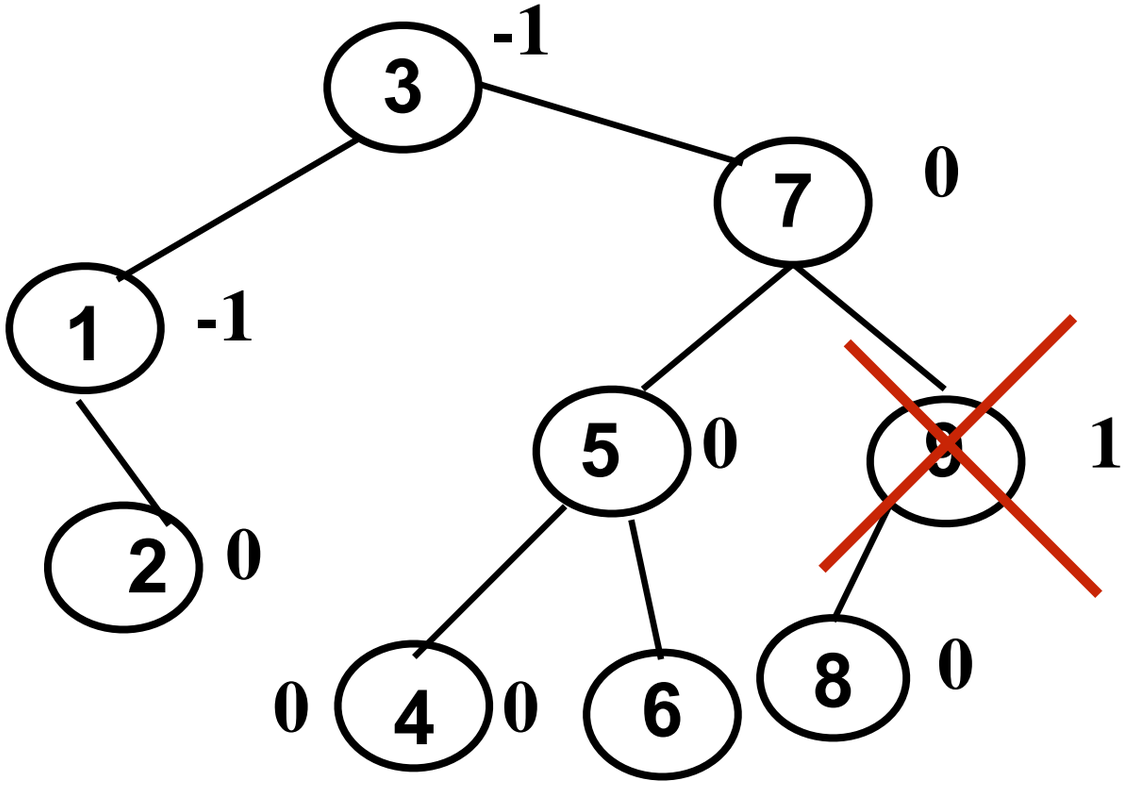
unico FB da aggiornare

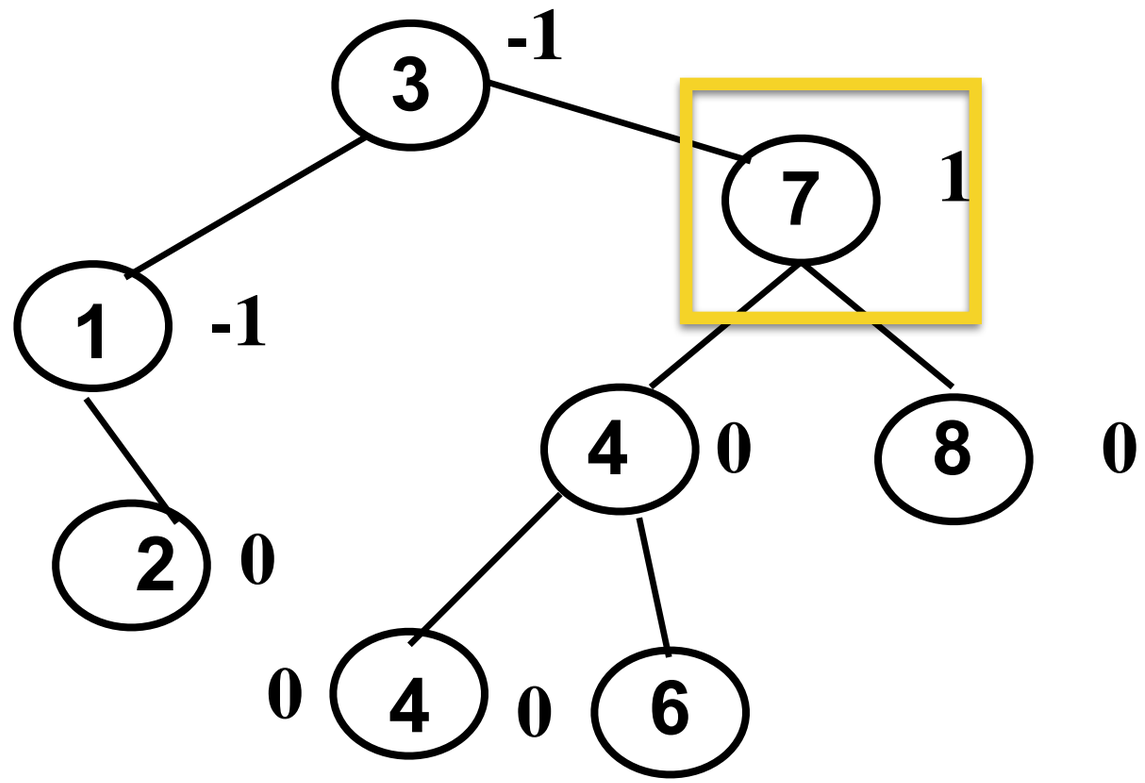
La cancellazione in un AVL

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

Esempio:

Cancellazione di un nodo con un solo figlio, 9





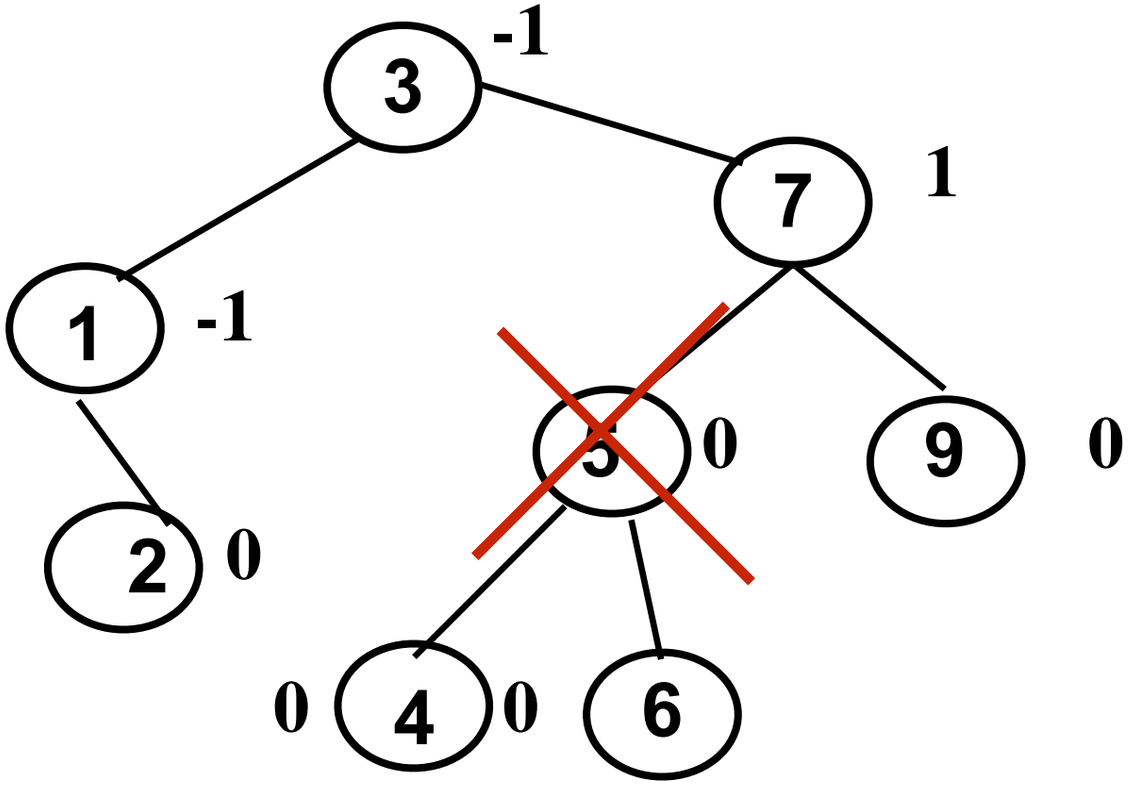
unico FB da aggiornare

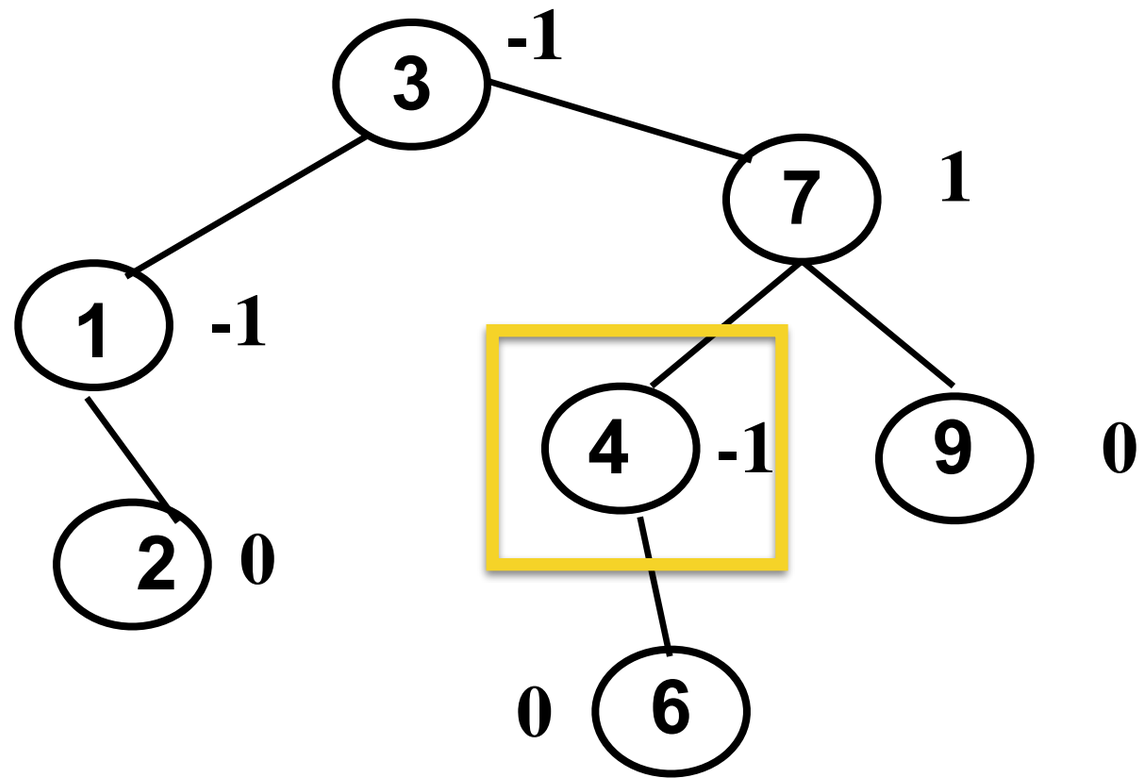
La cancellazione in un AVL

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

Esempio:

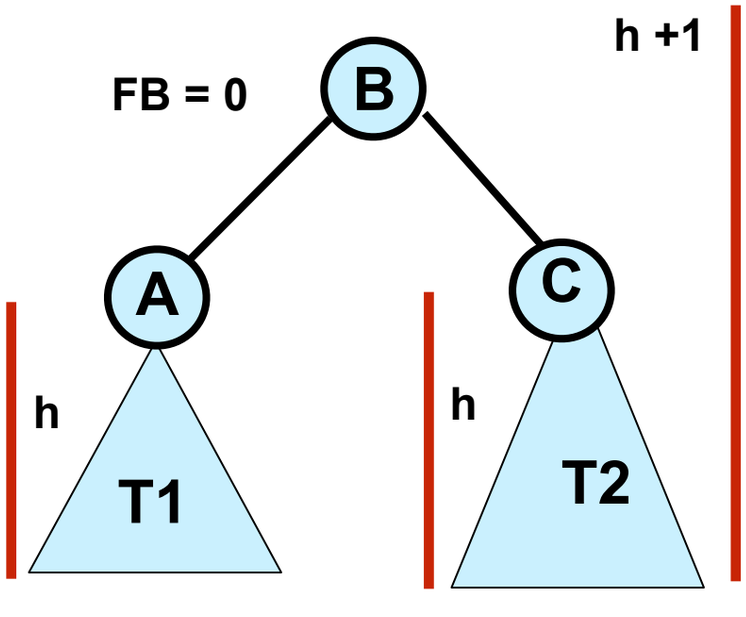
Cancellazione di un nodo con 2 figli, 5

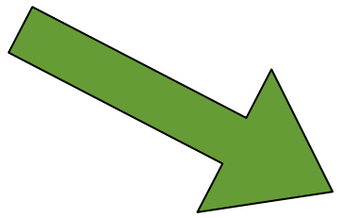




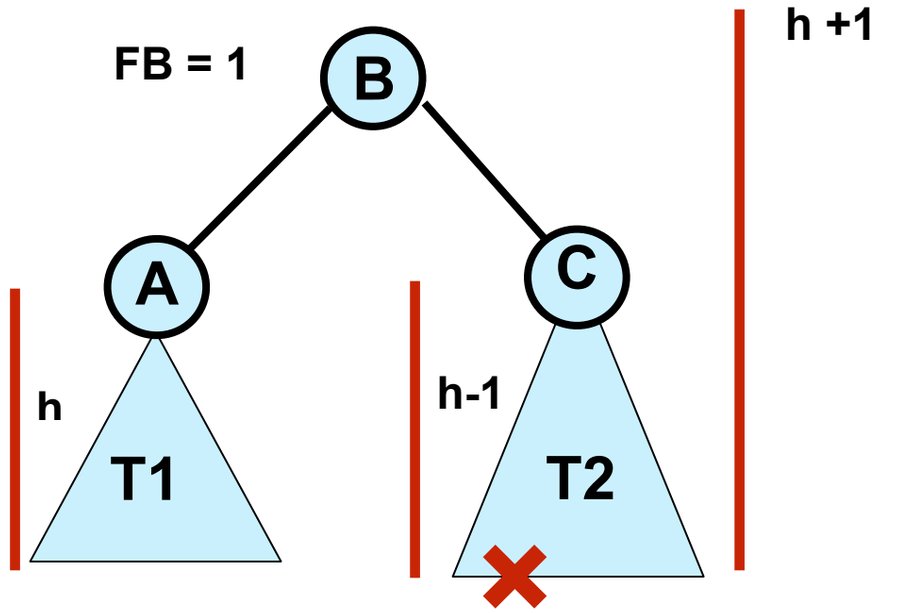
unico FB da aggiornare

Cancellazione: aggiornamento FB





cancellazione in T2



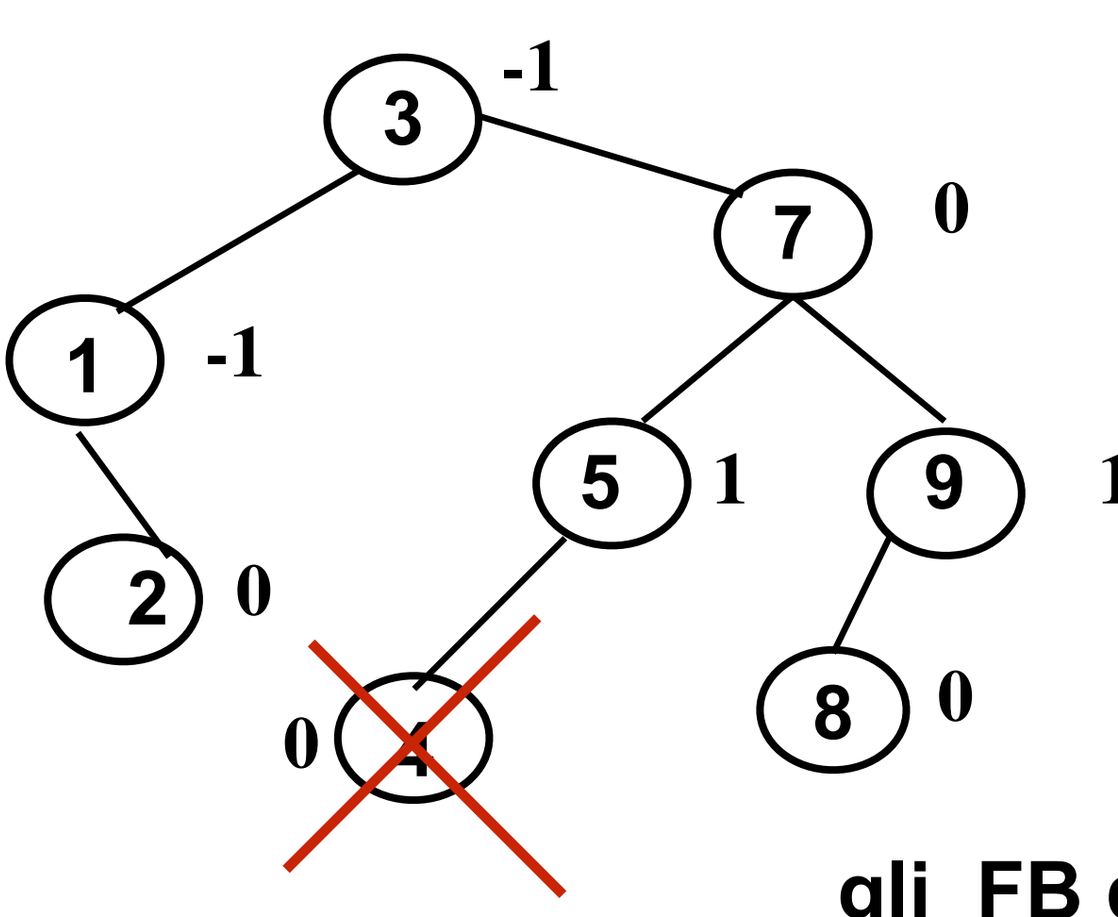
Se risalendo verso la radice il fattore di bilanciamento passa da 0 a 1 o da 0 a -1) si può fermare il processo di aggiornamento dei fattori di bilanciamento, perché vuol dire che la cancellazione non ha prodotto una diminuzione di altezza di tutto l'albero ma solo in un suo sotto albero.

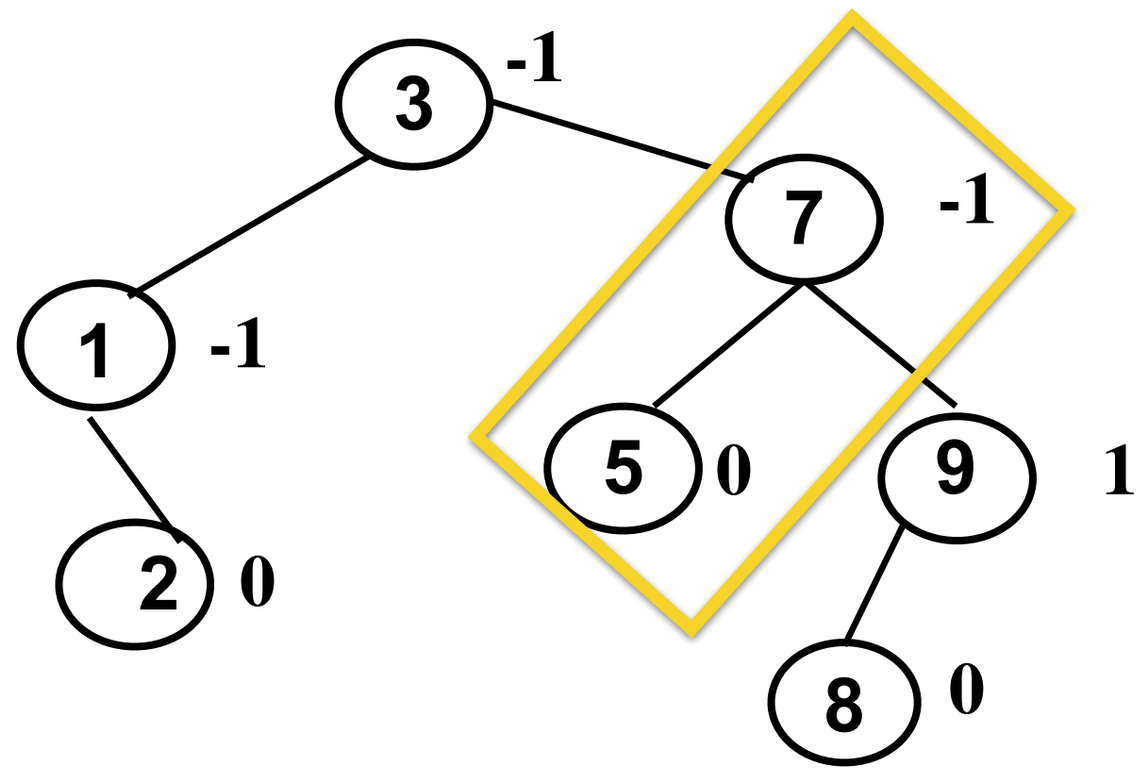
La cancellazione in un AVL

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

Esempio:

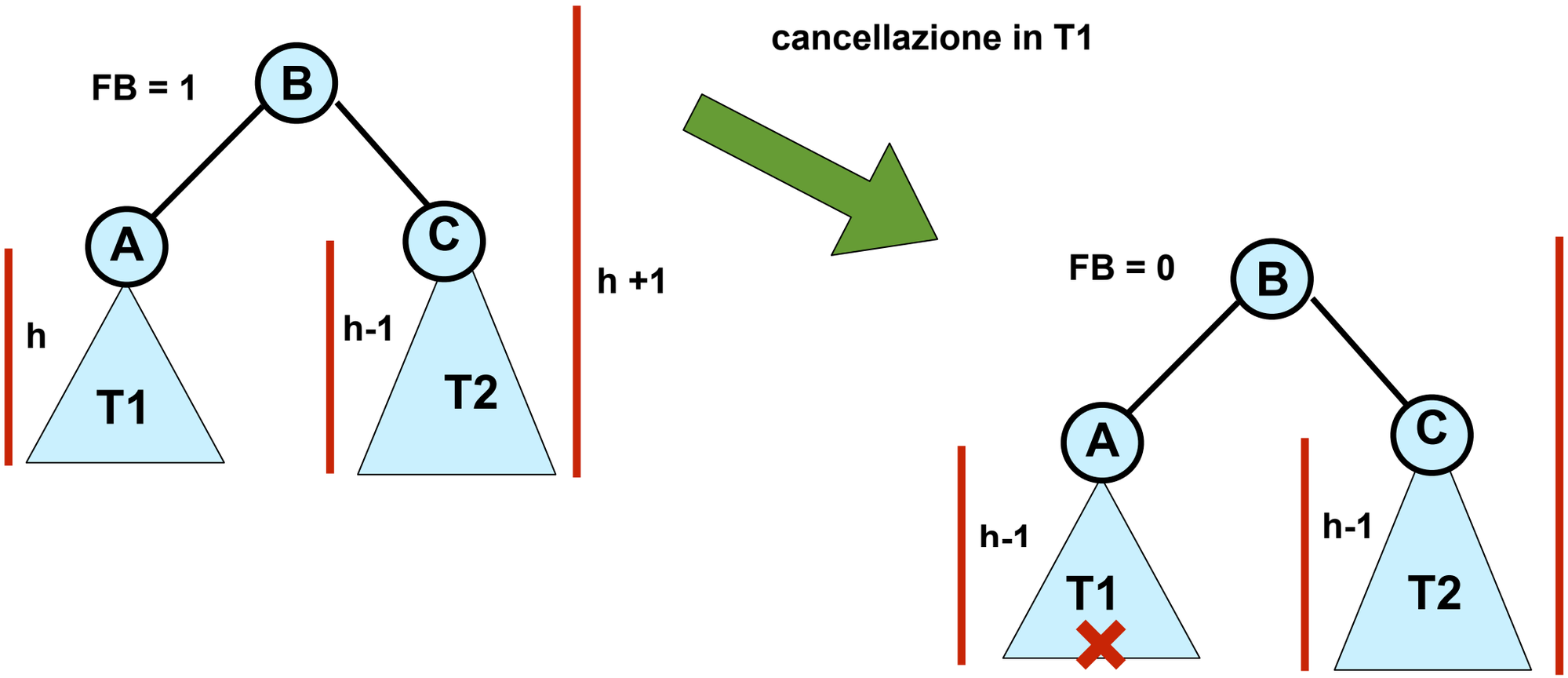
Cancellazione di una foglia, 4





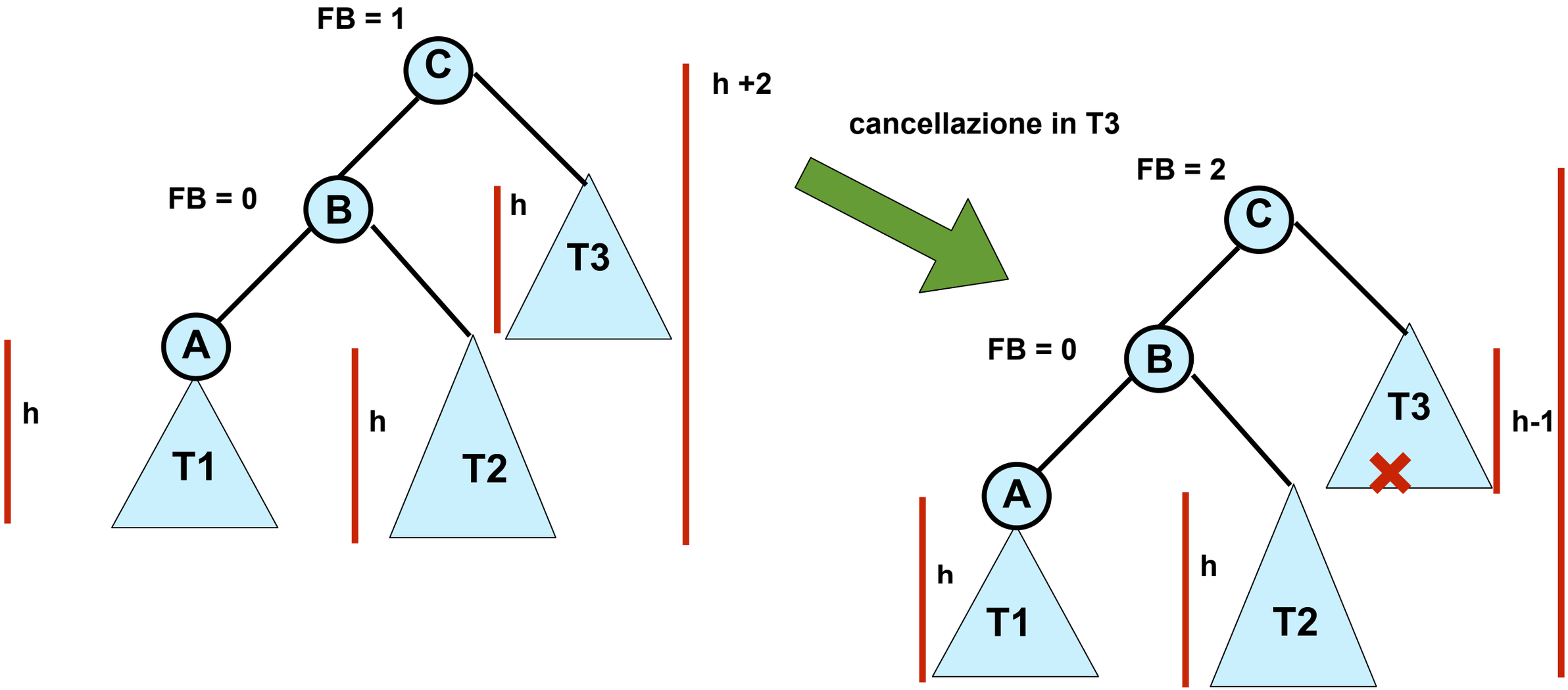
gli FB da aggiornare

Cancellazione: aggiornamento FB



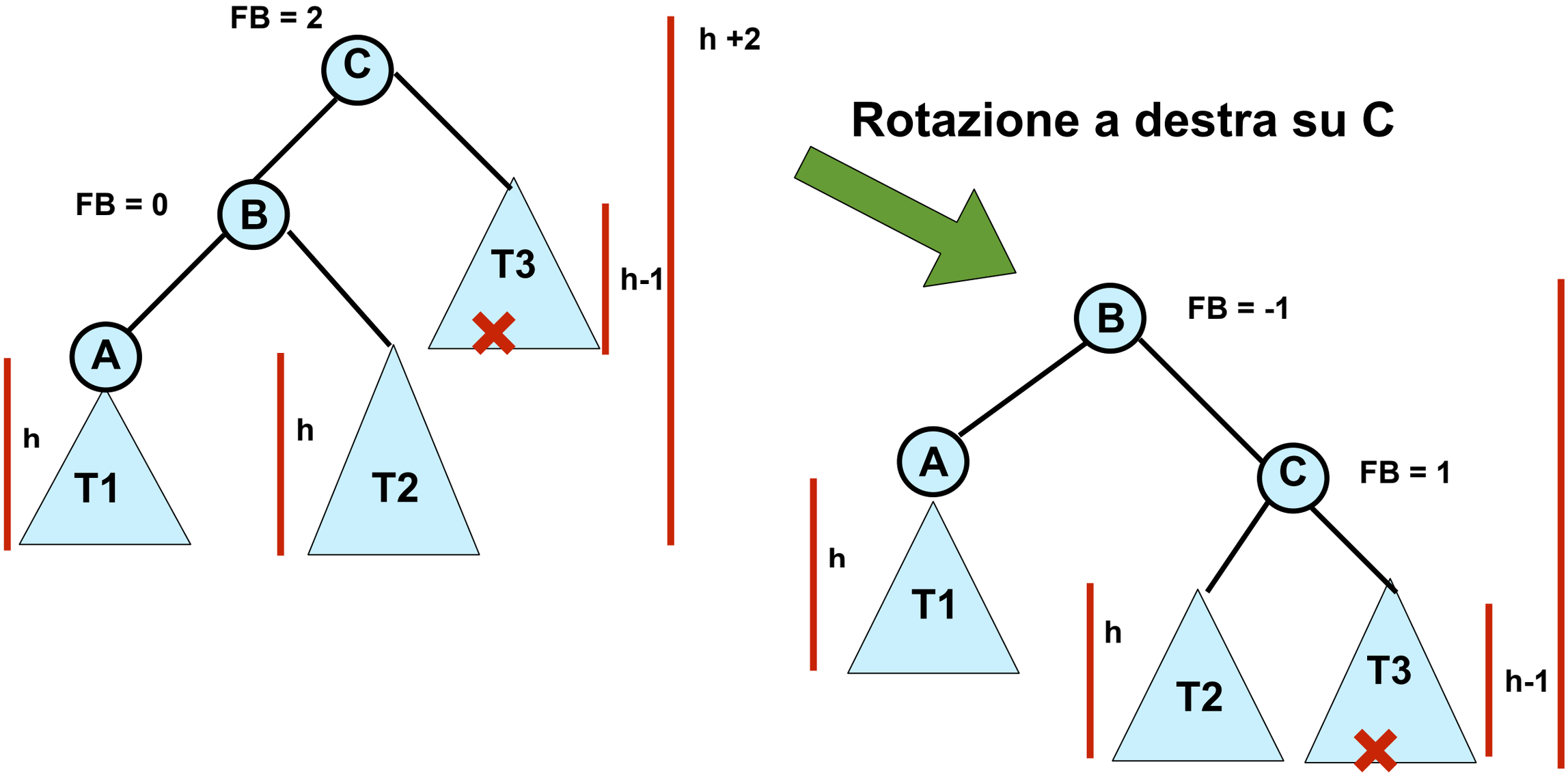
Se risalendo verso la radice il fattore di bilanciamento passa da 1 a 0 vuol dire che l'altezza è diminuita e si deve proseguire con il padre di B il processo di aggiornamento dei fattori di bilanciamento.

Cancellazione: CASO LEFT LEFT 1

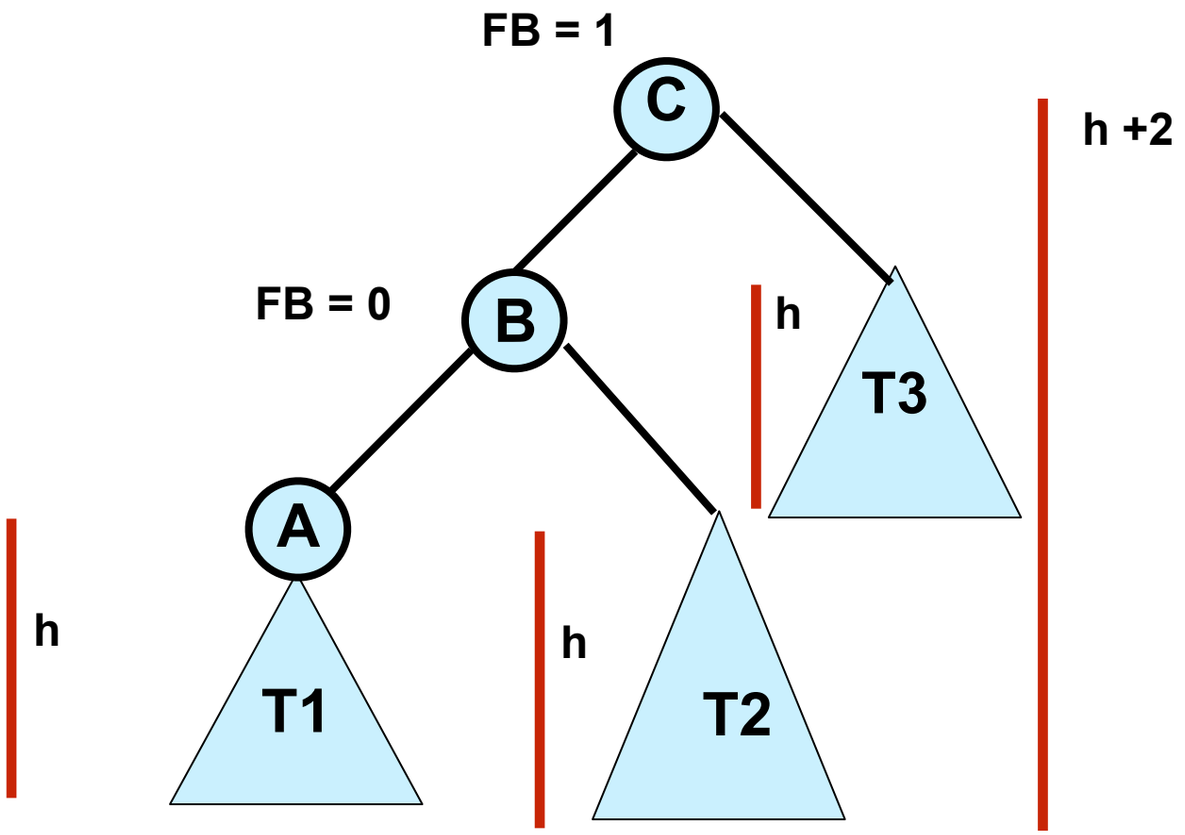


La cancellazione in T3 ne provoca una diminuzione di altezza, quindi nel nodo C il fattore di bilanciamento diventa illegale, anche se la sua altezza non cambia.
Si chiama LEFT LEFT perché corrisponde ad un inserimento in T1 che sbilanci il nodo C.

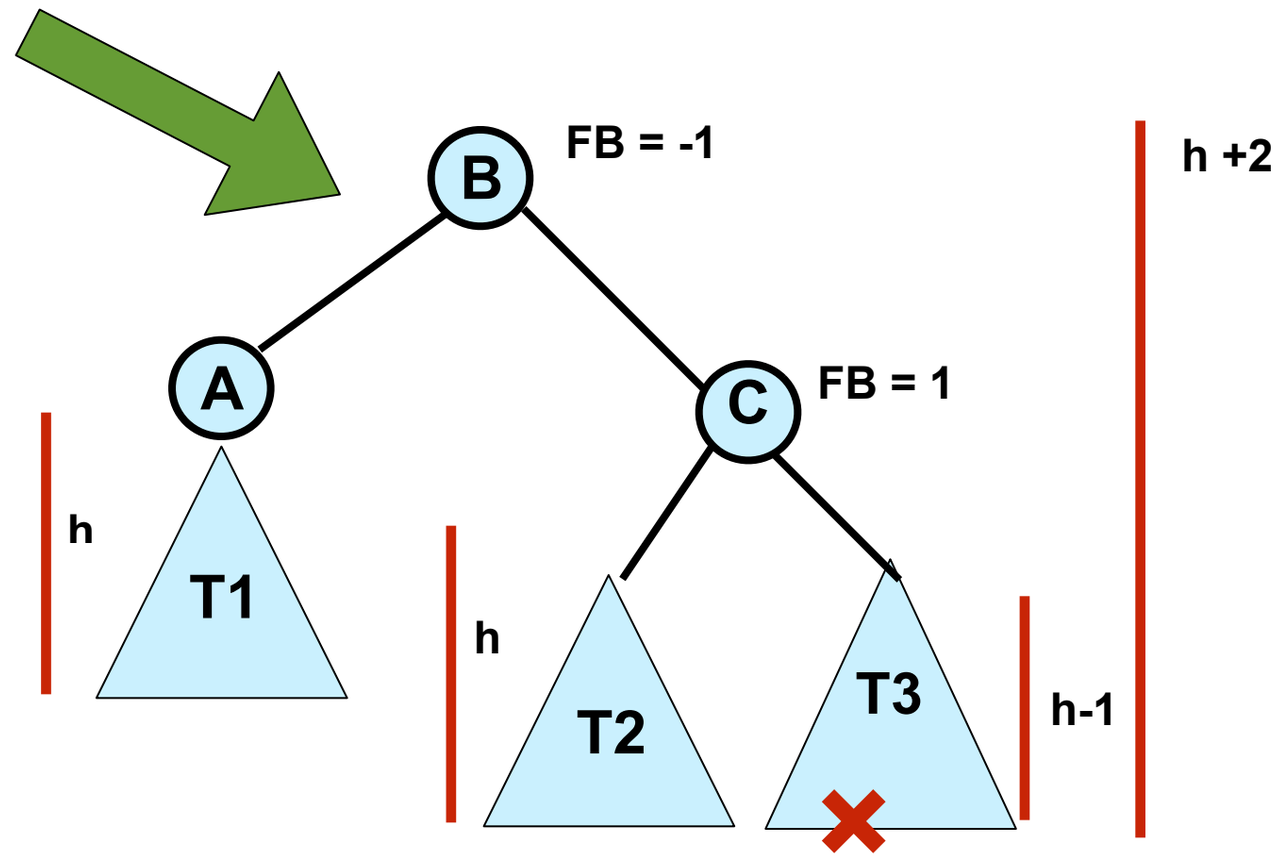
CASO LEFT LEFT 1- rotazione



CASO LEFT LEFT 1: conclusione



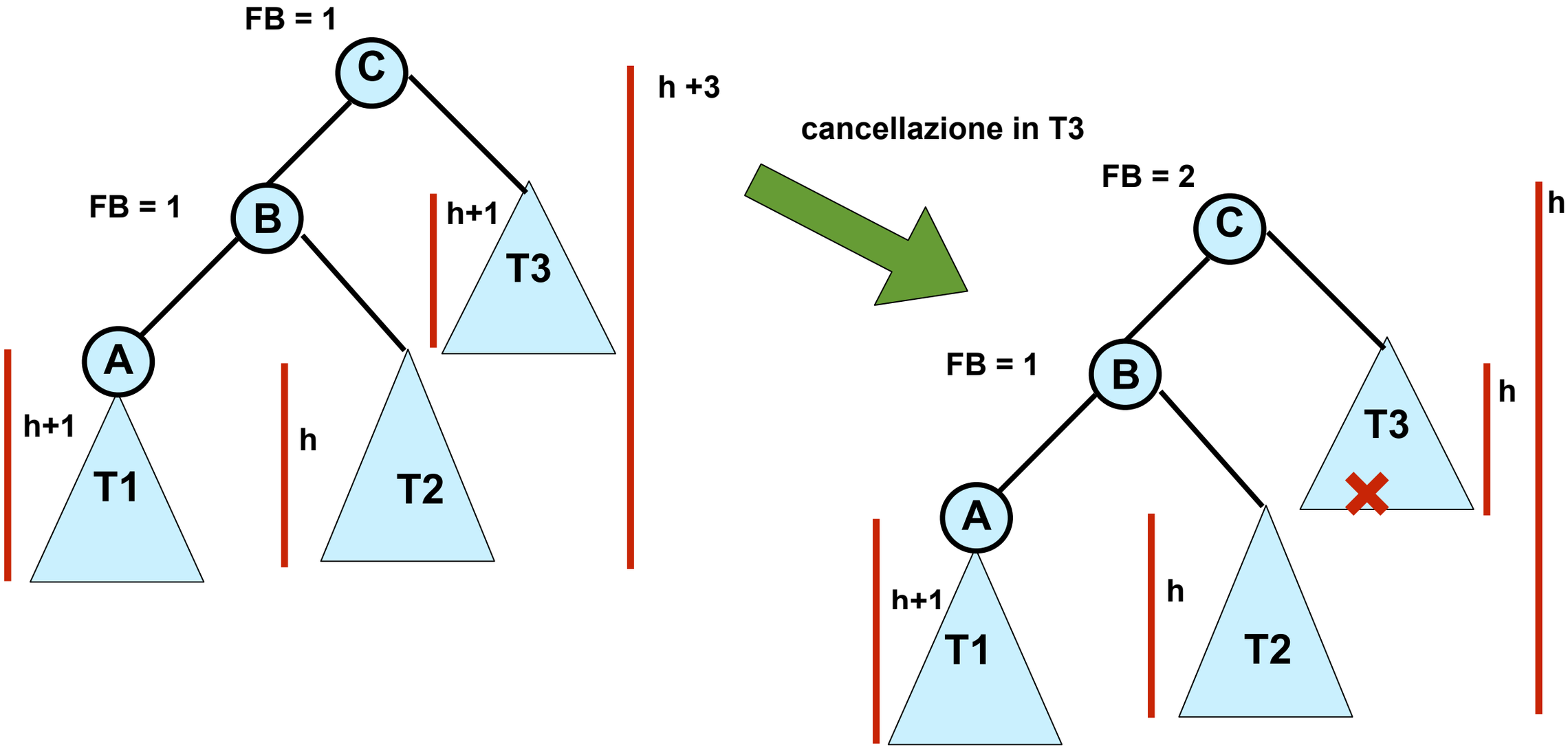
Cancellazione in T3 che ne diminuisce l'altezza e rotazione a destra su C



Dopo la rotazione l'altezza di B è quella del nodo C prima della cancellazione, quindi non è necessario risalire ancora verso la radice per ribilanciare l'albero!

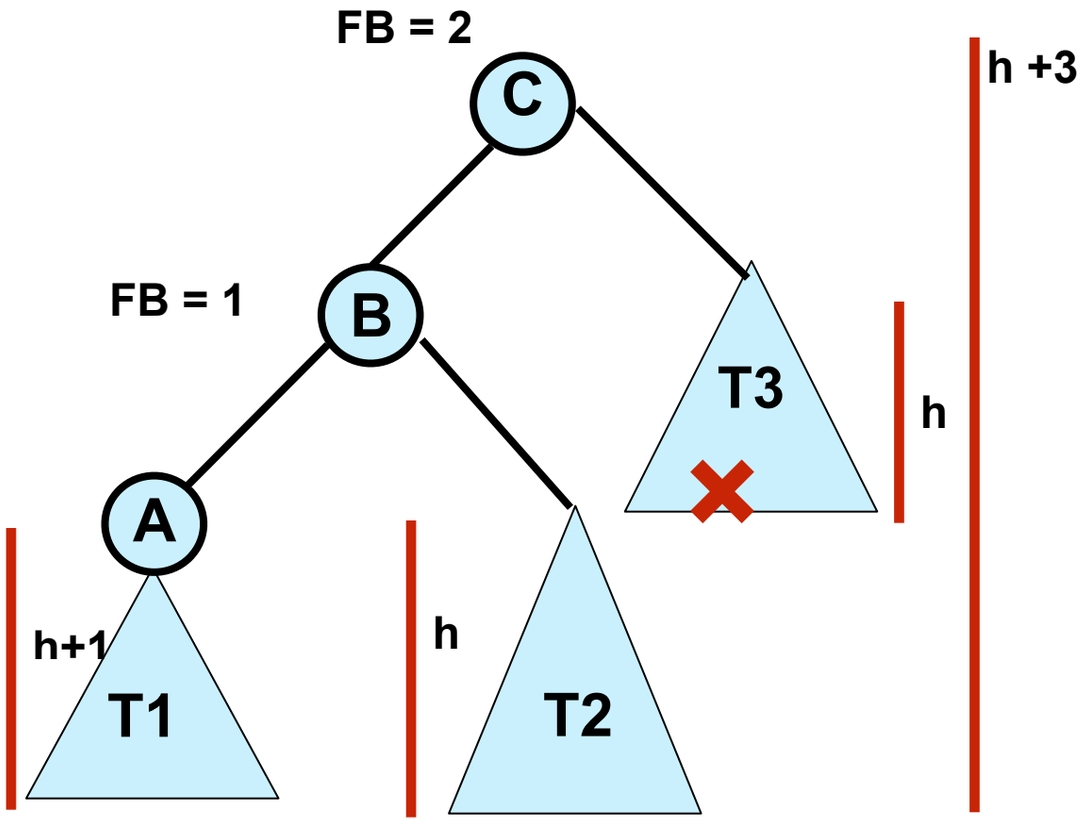
FB passa da 1 a -1

Cancellazione: CASO LEFT LEFT 2

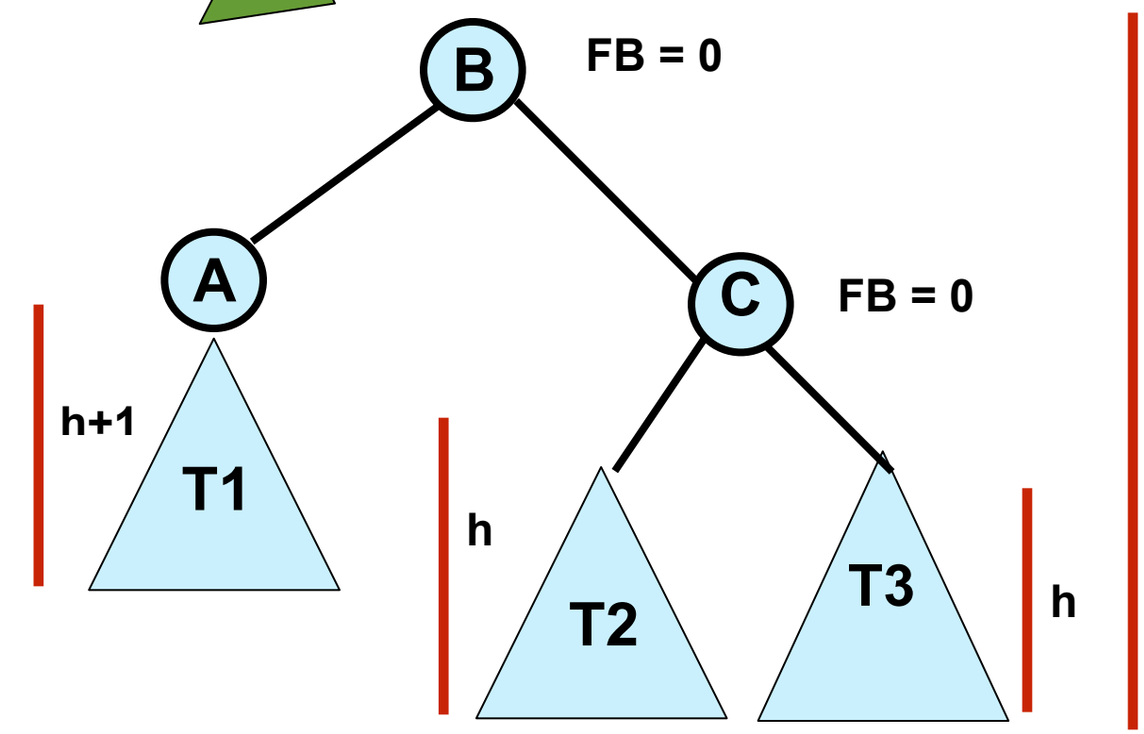


La cancellazione in T3 ne provoca una diminuzione di altezza, quindi nel nodo C il fattore di bilanciamento diventa illegale ma la sua altezza non cambia.

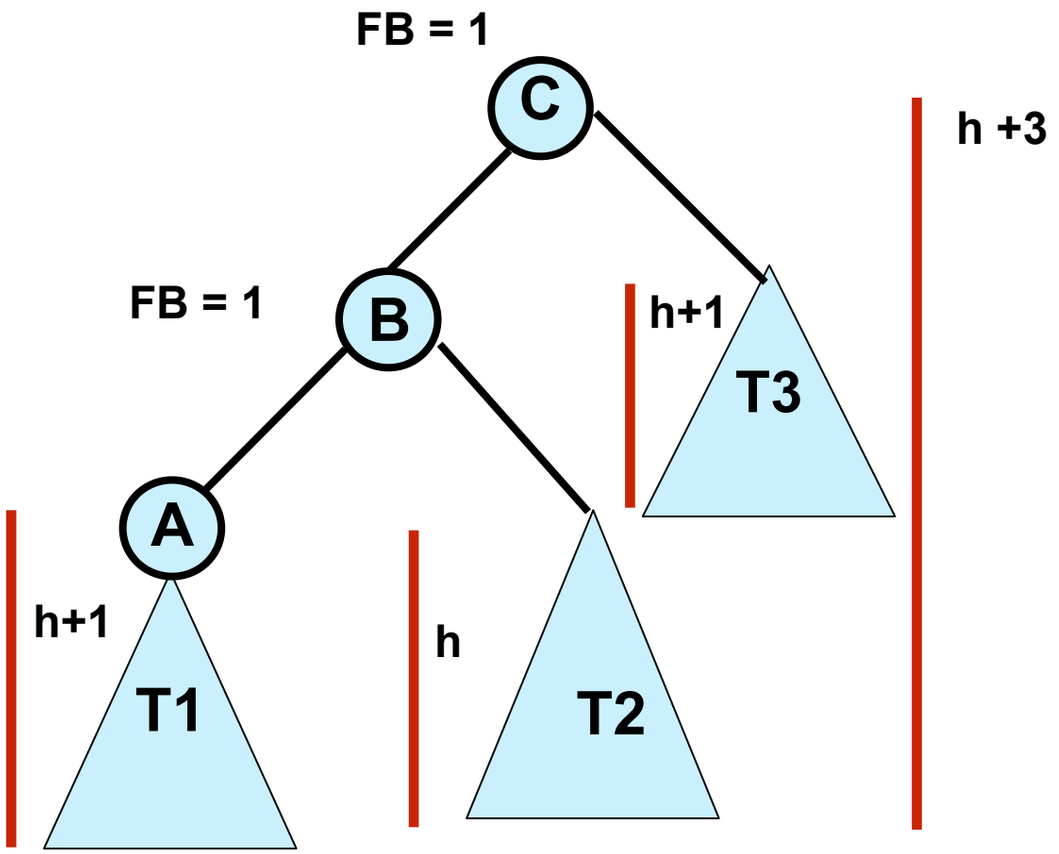
CASO LEFT LEFT 2 rotazione



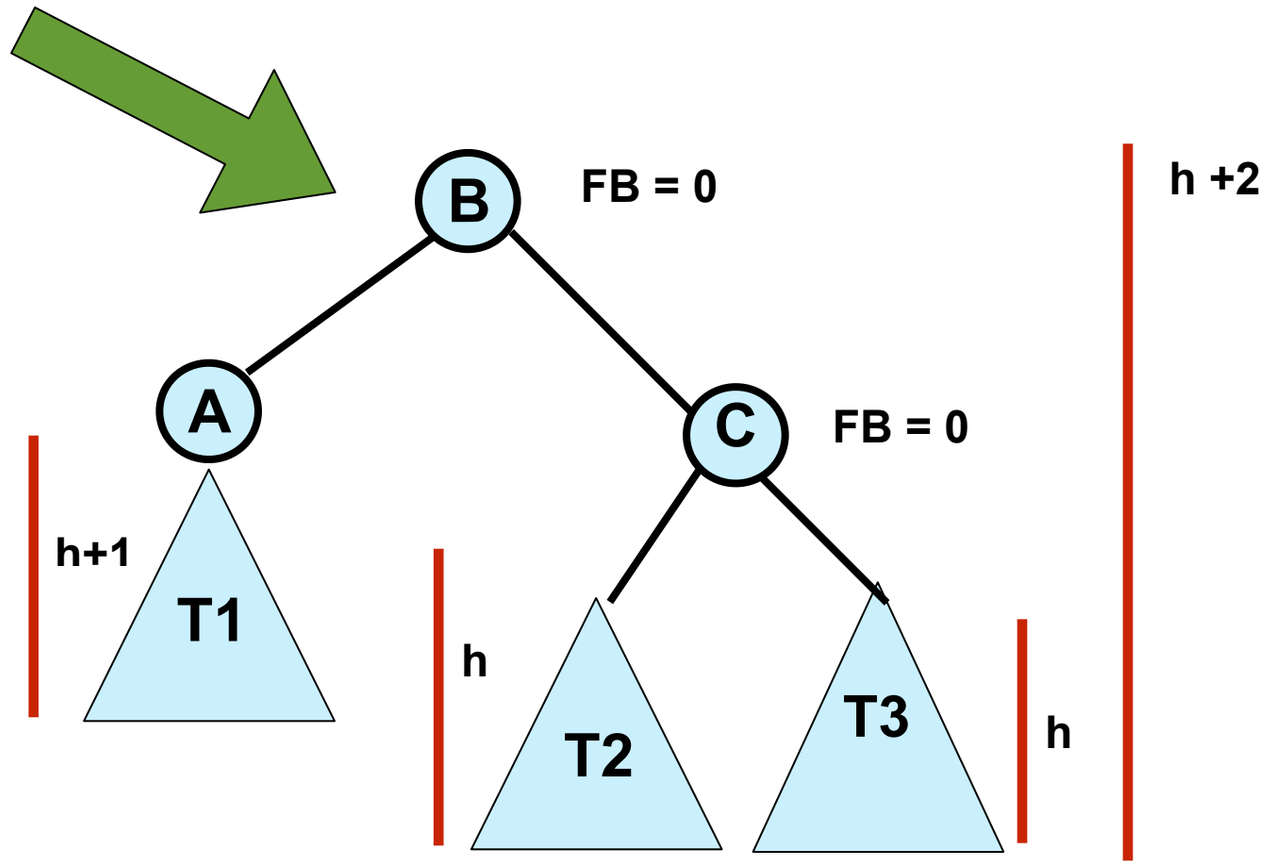
Rotazione a destra su C



CASO RIGHT 2 conclusione



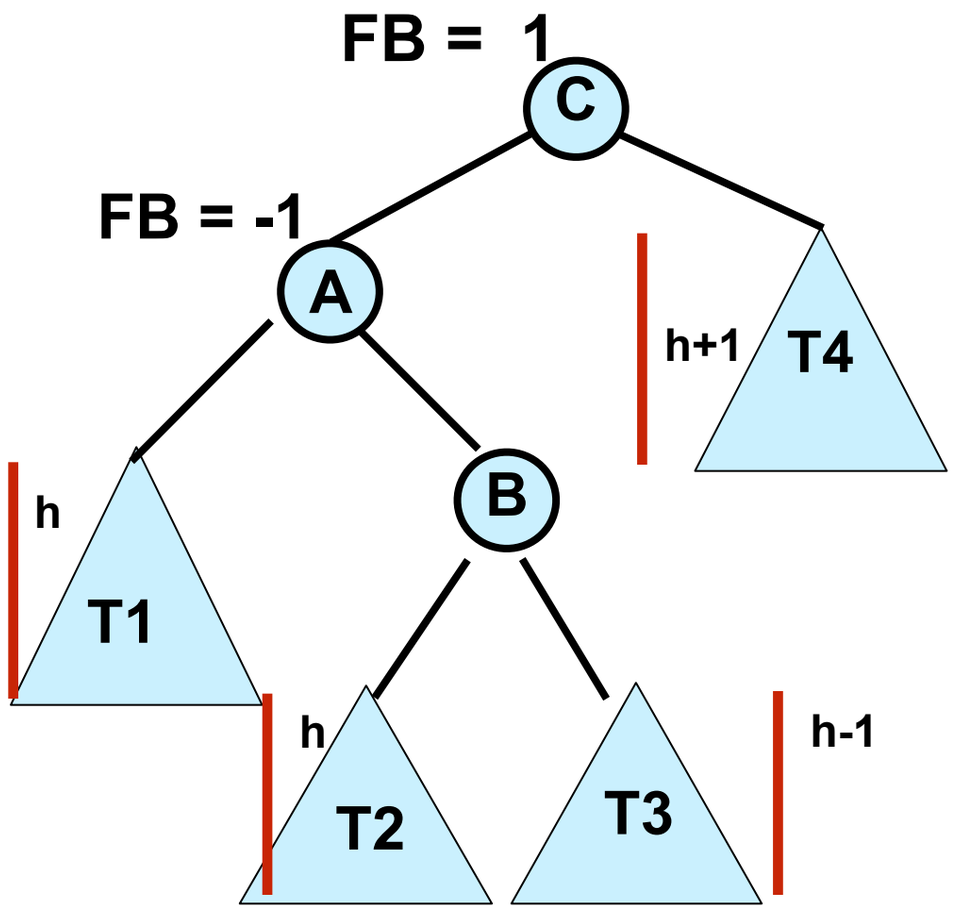
Cancellazione in T3 che ne diminuisce l'altezza e rotazione a destra su C



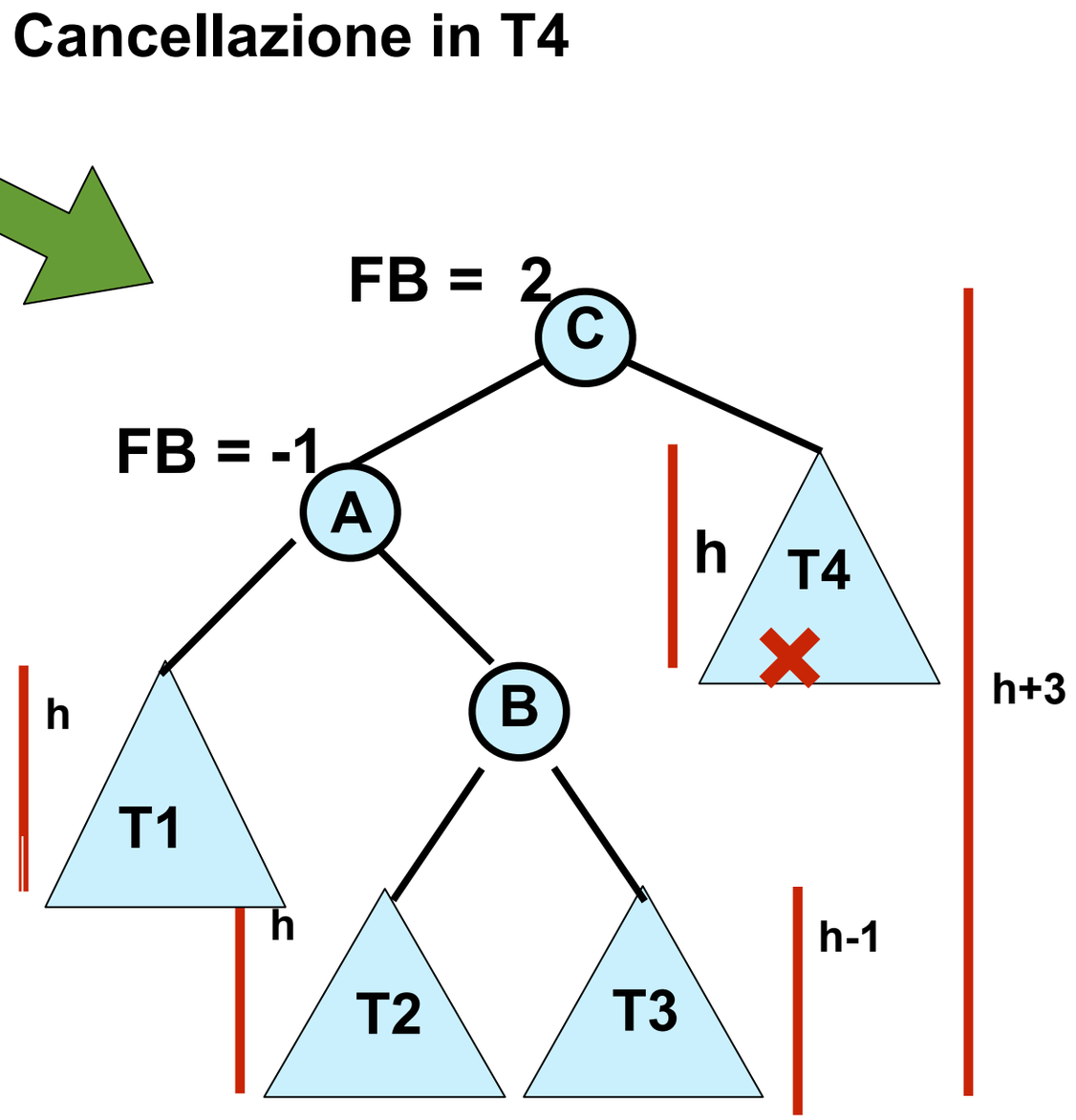
Dopo la rotazione l'altezza di B è minore di quella del nodo C prima dell'inserimento, quindi è necessario risalire ancora verso la radice per ribilanciare l'albero!

L'aggiornamento deve proseguire verso la radice perché si è passati da FB = 1 a un FB =0 per la radice del sotto albero il cui FB era diventato illegale.

CASO LEFT-RIGHT

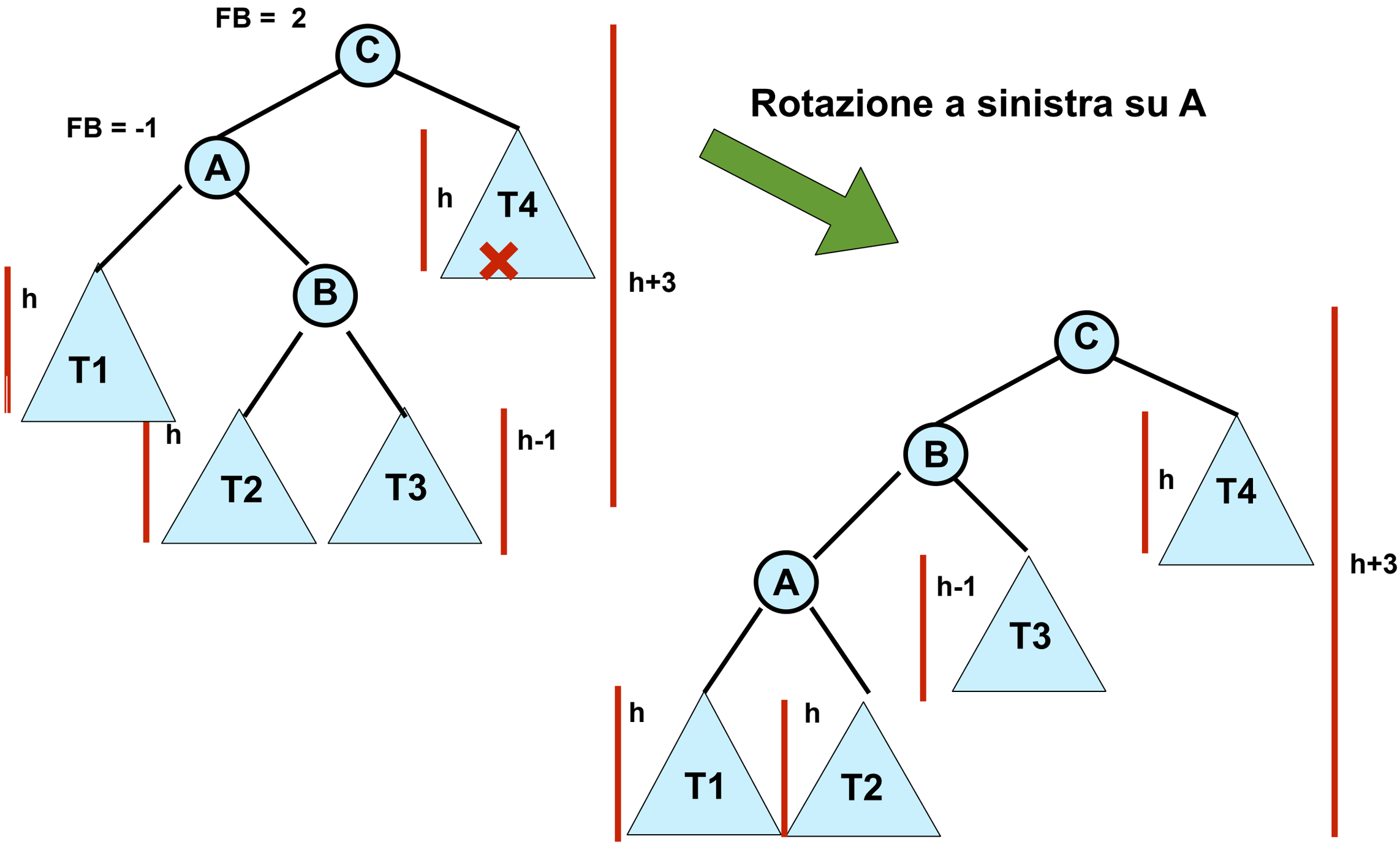


La cancellazione in T4 ne provoca la diminuzione dell' altezza, quindi nel nodo C il fattore di bilanciamento diventa illegale e suo figlio sinistro ha un BF di segno opposto, per cui servirà una doppia rotazione.

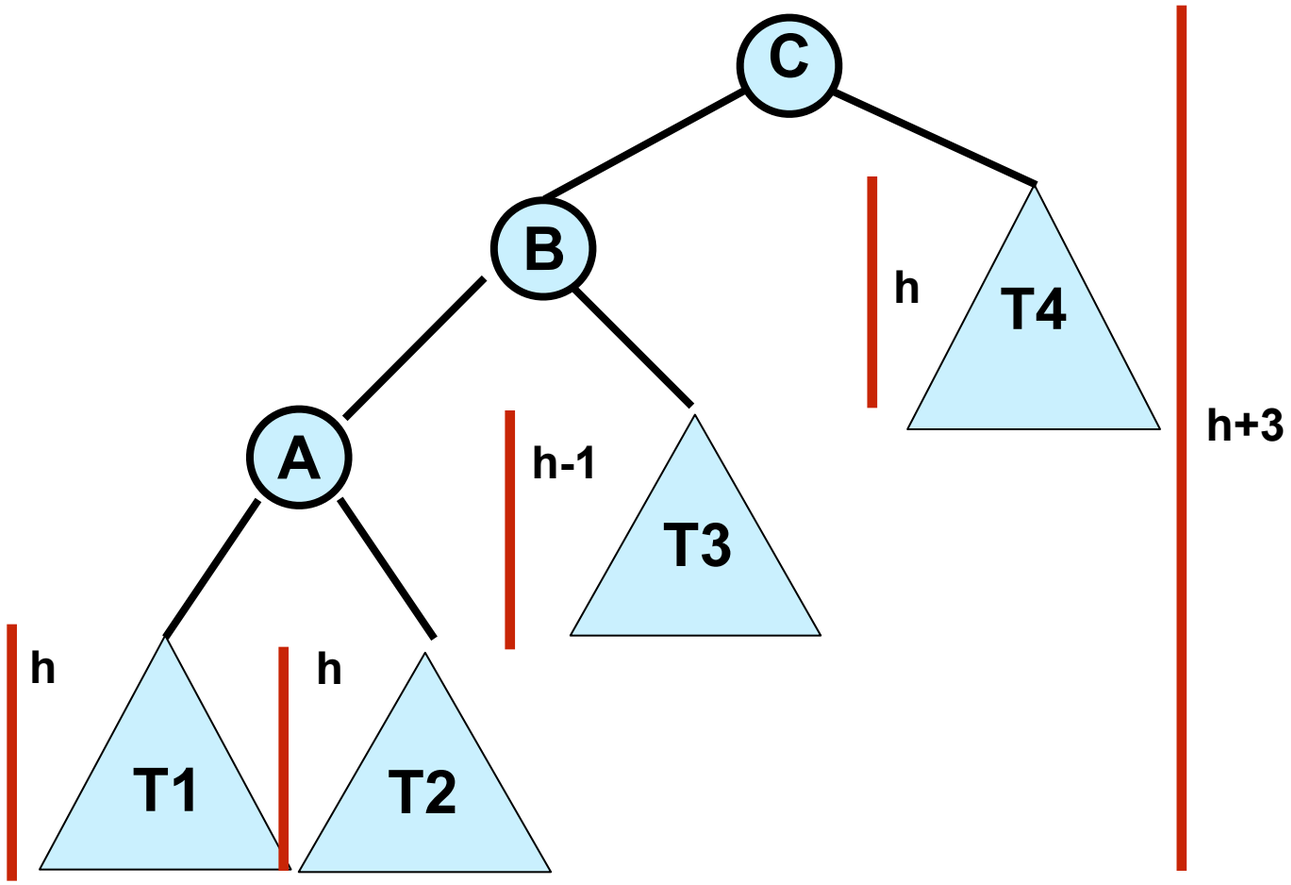


Questo caso corrisponde al caso LEFT-RIGHT dell'inserimento nel sotto albero radicato in B

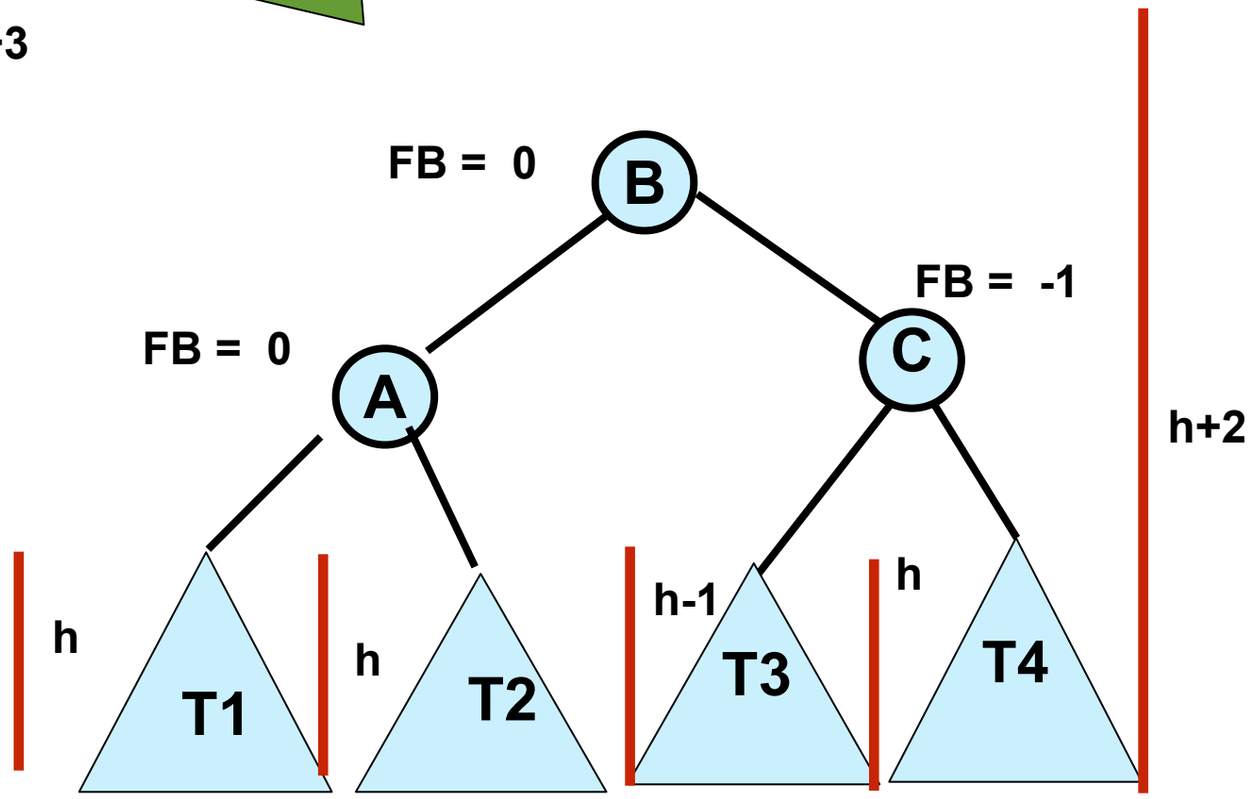
LA PRIMA ROTAZIONE



La seconda rotazione



Rotazione a destra su C



Anche in questo caso, dopo la seconda rotazione l'altezza di A è minore di quella del nodo C prima dell'inserimento, quindi è necessario risalire ancora verso la radice per ribilanciare l'albero!

L'aggiornamento deve proseguire infatti la nuova radice del sottoalbero ha FB =0 mentre il nodo C aveva FB = 1 prima della cancellazione, .

La cancellazione prevede quindi molti casi

Ai casi considerati si devono aggiungere i casi simmetrici. Spesso si fa uso della cosiddetta cancellazione “lazy”: si aggiunge ad ogni nodo un flag che segnala se il nodo è presente o no. Quindi la cancellazione è solo una ricerca con modifica del campo flag da “presente” a “assente”. Quando il numero dei nodi “assenti” supera una certa percentuale si ricostruisce l’albero inserendo solo i nodi presenti. Anche l’inserimento può avvantaggiarsi di questo approccio, se si vuole inserire un nodo “assente” basta cambiare il flag a “presente”

Pseudocodice

La funzione di inserimento viene modificata in modo tale che risalendo verso la radice si aggiornano i fattori di bilanciamento (o le altezze) nei nodi:
Se il nuovo nodo è stato inserito nel sotto albero sinistro il fb del padre (o la sua altezza) cresce di 1, altrimenti decresce di 1.
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.

Pseudocodice - ogni nodo ha fb

Ribilanciamento(T)
input: T è il riferimento all' 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
          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
          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)
```

Alberi binari di ricerca: confronto

Ben Pfaf, dip. Computer Science della Stanford Un. ha pubblicato nel 2004 un articolo in cui dà conto di una serie di risultati di confronto sperimentali sulle operazioni di ricerca, inserimento e cancellazione per i normali BST (ABR), gli AVL, i rosso-neri e gli "splay trees".

Qui riportiamo alcuni esiti dei confronti sperimentali tra rosso-neri, BST e AVL:

Tempo: i **rosso-neri sono i migliori quando le sequenze di elementi inseriti sono costruite a caso con occasionali inserimenti di sequenze di elementi ordinati, (la regola di bilanciamento per gli alberi rosso-neri è più permissiva di quella degli AVL quindi si fanno meno operazioni di ribilanciamento) se invece gli inserimenti di sequenze di elementi ordinati sono prevalenti allora sono gli AVL a comportarsi meglio.**

Se le sequenze di elementi inseriti sono costruite a caso a comportarsi meglio sono i semplici BST, ma se i dati sono inseriti in ordine allora i BST degenerano in liste.

Spazio: Per l'occupazione della memoria gli AVL richiedono 2 extra bits di memoria e i rosso-neri solo uno.

Se ci sono i puntatori al padre si deve aggiungere un campo puntatore a ogni nodo, e questa rappresentazione risulta la più efficiente in tempo per tutte le classi di alberi considerate.