Corso di laurea in Informatica
Introduzione agli Algoritmi
Lezioni in modalità mista o a distanza

Dizionari: Alberi Rosso-Neri

Tiziana Calamoneri

SAPIENZA
UNIVERSITÀ DI ROMA

Slides realizzate sulla base di quelle preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Bilanciamento dell'altezza (1)

Un ABR di altezza *h* contenente *n* nodi supporta le operazioni fondamentali dei dizionari in tempo *O(h)*, ma purtroppo non si può escludere che *h* sia *O(n)*, con conseguente degrado delle prestazioni.

Viceversa, tutte le operazioni restano efficienti se si riesce a garantire che l'altezza dell'albero sia piccola, in particolare sia *O(log n)*.

Per raggiungere tale scopo esistono varie tecniche, dette di ***bilanciamento***.

Bilanciamento dell'altezza (2)

Le tecniche di ***bilanciamento*** sono tutte basate sull'idea di riorganizzare la struttura dell'albero se essa, a seguito di un'operazione di inserimento o di eliminazione di un nodo, viola determinati requisiti.

In particolare, il requisito da controllare è che, per ciascun nodo dell'albero, l'altezza dei suoi due sottoalberi non sia "troppo differente".

Ciò che rende non banali queste tecniche è che si vuole aggiungere agli ABR una proprietà (il bilanciamento) senza peggiorare il costo computazionale delle operazioni.

In letteratura vi sono vari approcci. Noi ne descriveremo uno.

Alberi Rosso-Neri (1)

Un *albero rosso-nero (RB-albero)* è un ABR i cui nodi hanno un campo aggiuntivo, il ***colore***, che può essere solo rosso o nero.

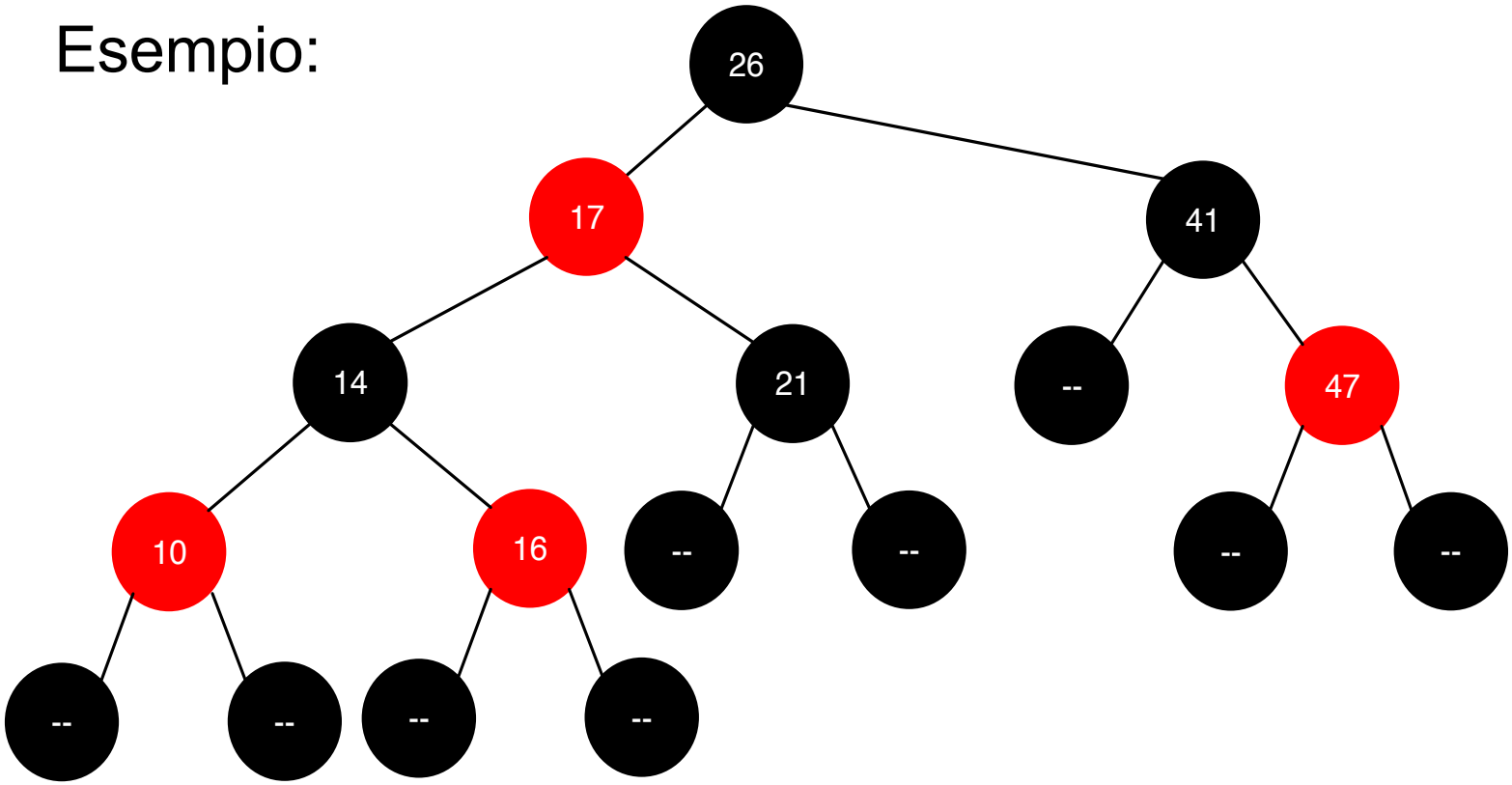
Alla struttura si aggiungono foglie fittizie (che non contengono chiavi) per fare in modo che tutti i nodi "veri" dell'albero abbiano ***esattamente due figli***.

Un RB-albero è un ABR che soddisfa le seguenti proprietà aggiuntive:

1. ciascun nodo è rosso o nero;
2. ciascuna foglia fittizia è nera;
3. se un nodo è rosso i suoi figli sono entrambi neri;
4. ogni cammino da un nodo a ciascuna delle foglie del suo sottoalbero contiene lo stesso numero di nodi neri.
5. La radice è sempre nera

Alberi Rosso-Neri (2)

Esempio:



Nota: per non sprecare spazio in memoria. l'insieme delle foglie fittizie può essere sostituito da un unico oggetto, cui puntano tutti i puntatori che indicano una foglia fittizia (sentinella).

Alberi Rosso-Neri (3)

Proprietà: nessun cammino dalla radice ad una foglia può essere lungo più del doppio di un cammino dalla radice ad una qualunque altra foglia.

Dim.

- il numero di nodi neri è il medesimo lungo tutti i cammini dalla radice ad una qualunque foglia (proprietà 4);
- il numero di nodi rossi lungo un cammino non può essere maggiore del numero di nodi neri lungo lo stesso cammino (proprietà 3);
- un cammino che contiene il massimo numero possibile di nodi rossi non può essere lungo più del doppio di un cammino composto solo di nodi neri.

CVD

Alberi Rosso-Neri (4)

La ***b-altezza*** di un nodo ***x*** (***black height*** di *x*, ***bh(x)***) è il numero di nodi neri sui cammini dal nodo x (non incluso) alle foglie sue discendenti (è uguale per tutti i cammini, proprietà 4).

La *b-altezza* di un RB-albero è la b-altezza della sua radice.

NOTA: se un albero rispetta le proprietà 1-4 ma ha la radice rossa (e quindi non rispetta la proprietà 5), basta cambiare il colore della radice in nero per ottenere un RB-albero valido. La sua b-altezza risulterà aumentata di 1.

Alberi Rosso-Neri (5)

Lemma

Il sottoalbero radicato in un qualsiasi nodo *x* contiene almeno $2^{bh(x)}-1$ nodi interni.

Dimostrazione

Ragioniamo per induzione sulla distanza di *x* dalle foglie, sia essa *d(x)*.

Se $d(x) = 0$, allora *x* è una foglia per cui $bh(x) = 0$.

Quindi il suo sottoalbero contiene almeno

$$2^{bh(x)} - 1 = 2^0 - 1 = 1 - 1 = 0$$

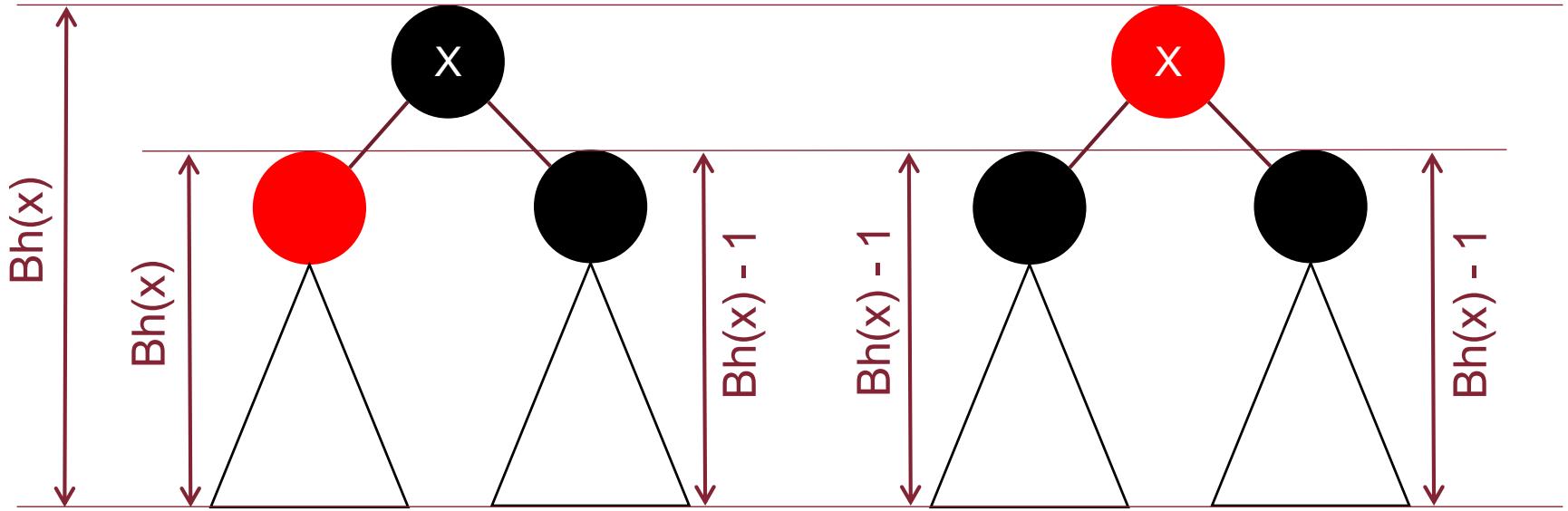
nodi interni.

Alberi Rosso-Neri (6)

Dimostrazione (continua)

Sia *x* un nodo interno, con *d(x) > 0*.

I suoi figli hanno b-altezza pari a *bh(x)* o a *bh(x)-1*, a seconda che essi siano rossi o neri:



Alberi Rosso-Neri (7)

Dimostrazione (continua)

Dato che i figli di *x* sono a distanza *d(x) – 1* dalle foglie, per essi vale l'ipotesi induttiva.

Quindi il sottoalbero di ciascuno dei due figli contiene almeno $2^{bh(x)-1}-1$ nodi interni.

Da ciò segue che i nodi interni del sottoalbero radicato in *x* sono almeno

$$2(2^{bh(x)-1}-1)+1=2^{bh(x)}-1.$$

CVD

Alberi Rosso-Neri (8)

Teorema

Un RB-albero con *n* nodi interni ha altezza *h*:

$$h \leq 2 \log(n + 1)$$

Dimostrazione

Sia *h* l'altezza dell'RB-albero ed *r* la sua radice.

Sappiamo già che $h \leq 2bh(r)$, ossia $bh(r) \geq h/2$.

Dal lemma precedente, il numero di nodi interni dell'RB-albero, pari al numero di nodi interni nel sottoalbero radicato in *r*, è:

$$n \geq 2^{bh(r)} - 1 \geq 2^{h/2} - 1$$

da cui

$$n + 1 \geq 2^{h/2} \text{ cioè } 2 \log(n + 1) \geq h.$$

CVD

Operazioni su RB-alberi (1)

Il teorema precedente garantisce che le operazioni di:

- ricerca di una chiave
- ricerca del massimo o del minimo
- ricerca del predecessore o del successore

siano tutte eseguite con un costo computazionale *O(log n)*.

Operazioni su RB-alberi (2)

Discorso a parte va fatto invece per gli inserimenti e le cancellazioni.

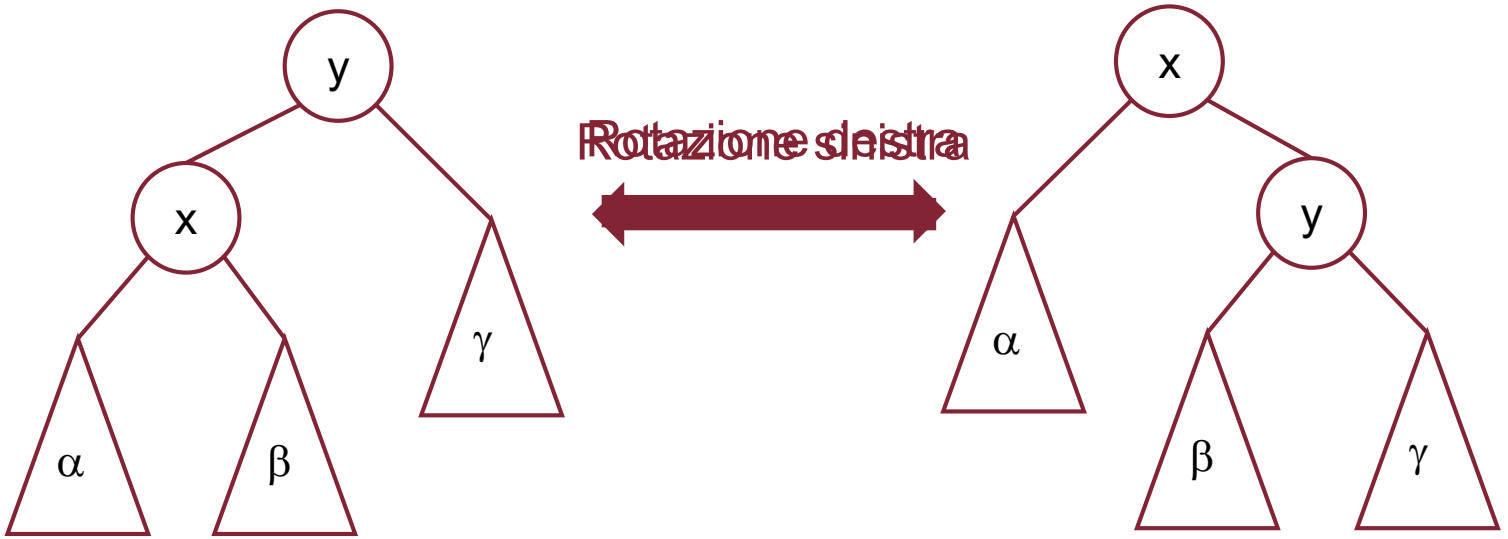
Infatti, l'esigenza di mantenere le proprietà di RB-albero implica che, dopo un inserimento o una cancellazione, la struttura dell'albero possa dover essere riaggiustata, in termini di:

- colori assegnati ai nodi;
- struttura dei puntatori (ossia, collocazione dei nodi nell'albero).

A tal fine sono definite apposite operazioni, dette ***rotazioni***, che permettono di ripristinare in tempo $O(\log n)$ le proprietà del RB-albero dopo un inserimento o una cancellazione.

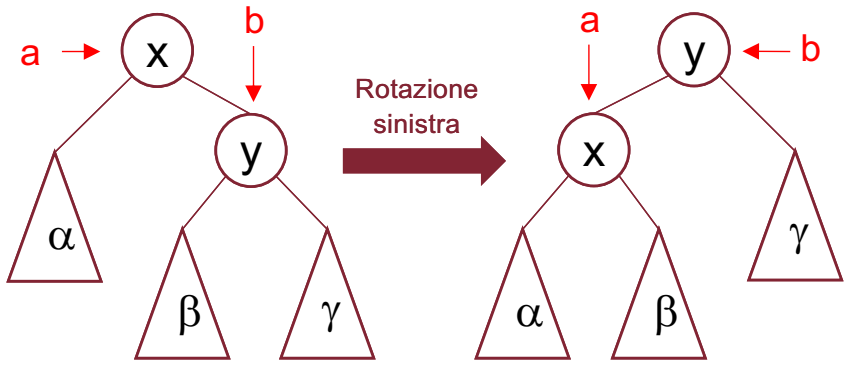
Rotazioni (1)

Le rotazioni possono essere ***destre*** o ***sinistre*** e sono operazioni locali che non modificano l'ordinamento delle chiavi secondo la visita in-ordine.



Rotazioni (2)

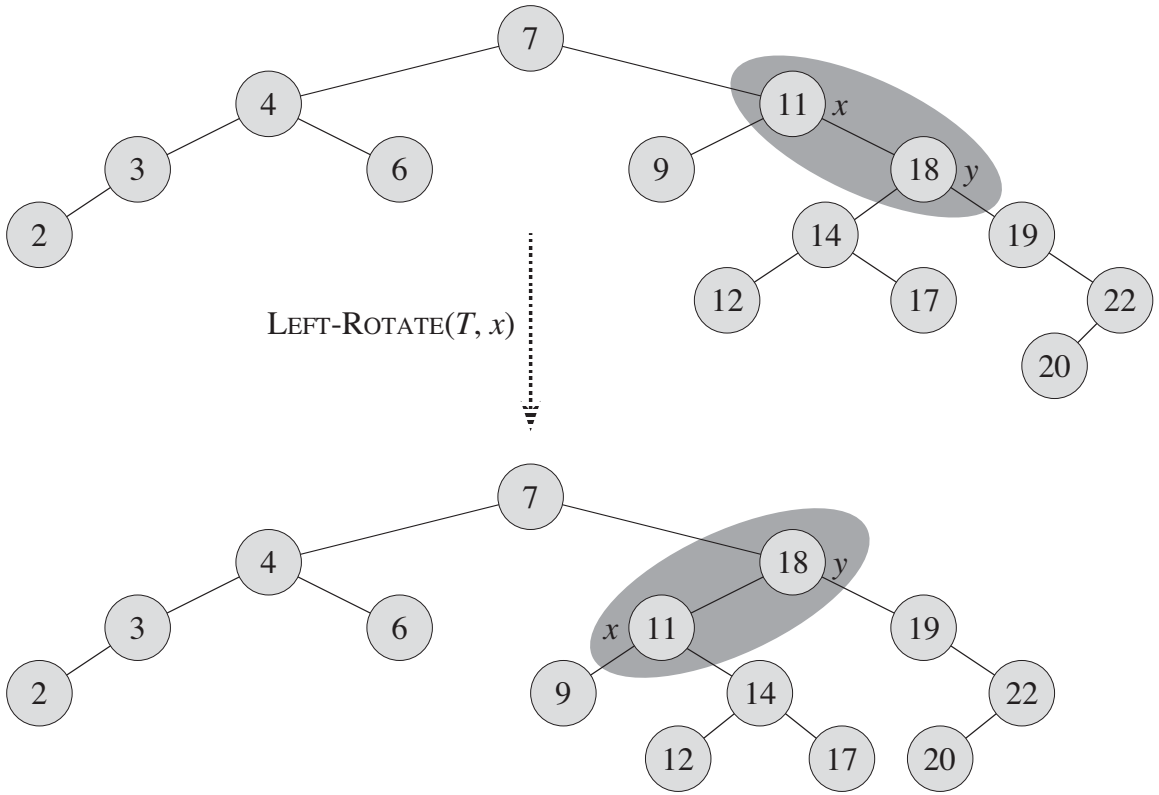
```
def RBA_rotaz_sinistra (p,a): #a=punt. al nodo su cui si ruota
    b = a->right
    a->right = b->left
    if a->right ≠ None:
       a->right->parent = a
    b->left = a
    b->parent = a->parent
    if a->parent==None:
       p = b
    else:
         if a==a->parent->left:
           a->parent->left = b
         else:
           a->parent->right = b
    a->parent = b
    return p
```



Costo computazionale: *Θ(1)*.

(La rotazione a destra è analoga)

Rotazioni (3)

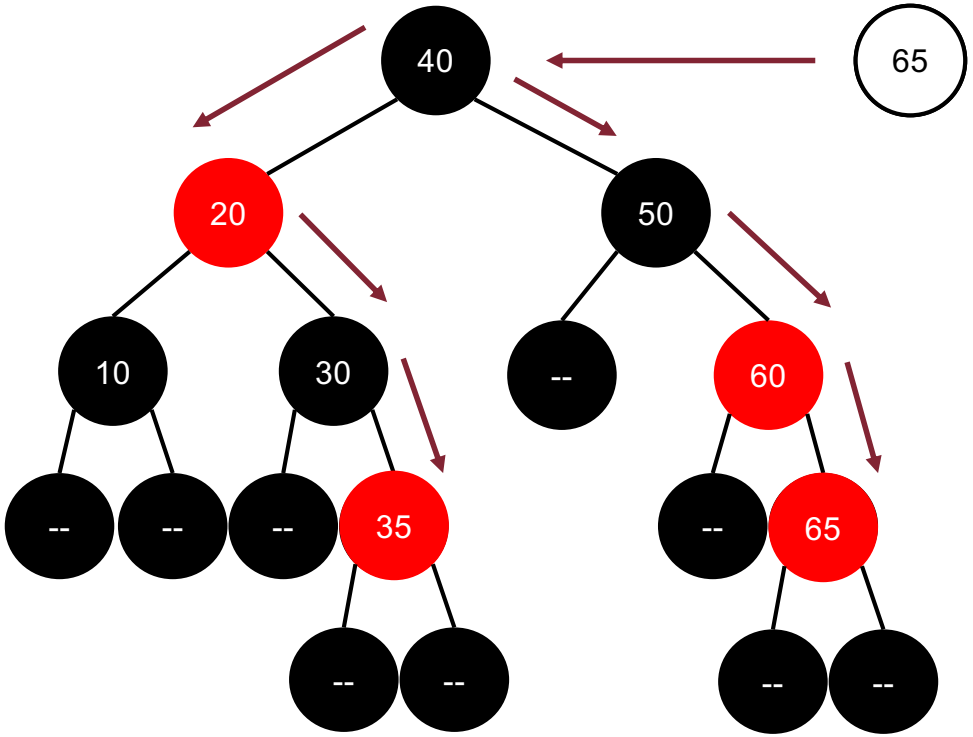


Nota: la visita in-order prima e dopo la rotazione produce lo stesso elenco di chiavi!

Inserimento (1)

Passo Preliminare: Si inserisce l'elemento seguendo le regole dell'inserimento in un ABR, attribuendo sempre colore rosso al nuovo nodo.

Esempio:



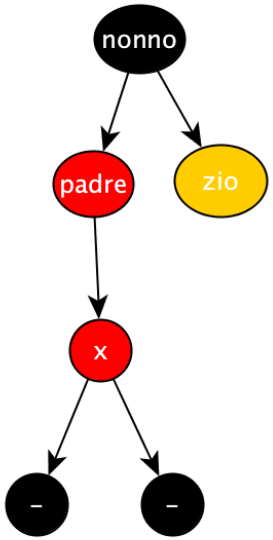
Inserimento (2)

- Le regole 1. (ciascun nodo è rosso o nero) e 2. (le foglie fittizie sono nere) si mantengono sempre vere nella struttura.
- Anche la regola 4. (ogni cammino da un nodo a ciascuna foglia ha lo stesso numero di nodi neri) rimane vera dopo un inserimento, visto che il nuovo nodo è sempre colorato di rosso.
- Le uniche regole che possono essere infrante sono:
  - La 5. (la radice è sempre nera) … si aggiusta facilmente…
  - la 3. (entrambi i figli di un nodo rosso sono neri), infatti il nuovo nodo (rosso) potrebbe essere inserito come figlio di un nodo rosso… ???

Inserimento (3)

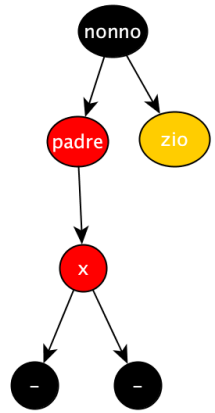
Passo di aggiustamento: Va eseguito solo se il nuovo nodo è stato inserito come figlio (rosso) di un nodo rosso.

Situazione corrente:



- il nonno esiste (perché un nodo rosso non può essere radice)
- il nonno è nero (perché un nodo rosso non può avere avere un figlio rosso)
- lo zio esiste (perché ogni nodo interno ha sempre due figli)

Inserimento (4)



4 possibili eventualità:

Caso 0: il nodo inserito è radice

Caso 1: il nodo inserito è figlio di un nodo rosso e lo zio è rosso.

Caso 2: il nodo inserito è figlio destro di un nodo rosso e lo zio è nero.

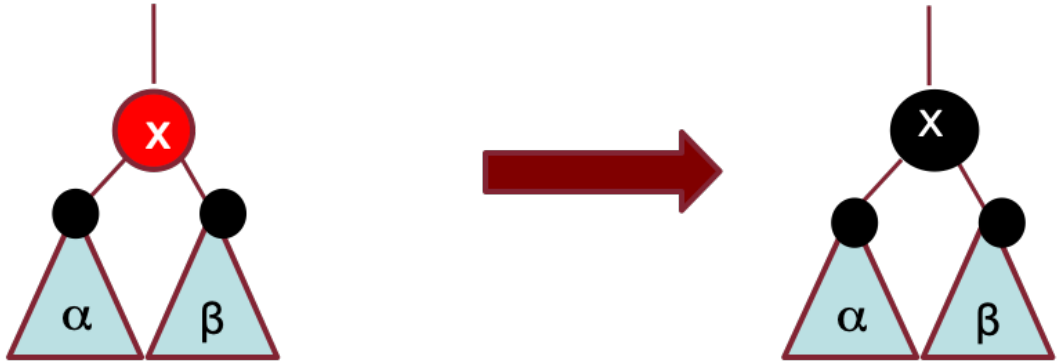
Caso 3: il nodo inserito è figlio sinistro di un nodo rosso e lo zio è nero.

N.B. nei casi 1-3, il nonno del nuovo nodo è nero, poiché l'unico punto dell'albero in cui la regola 3. non è rispettata è quello che stiamo considerando.

Inserimento (5)

Caso 0: Il nodo *x* che crea la violazione è radice.

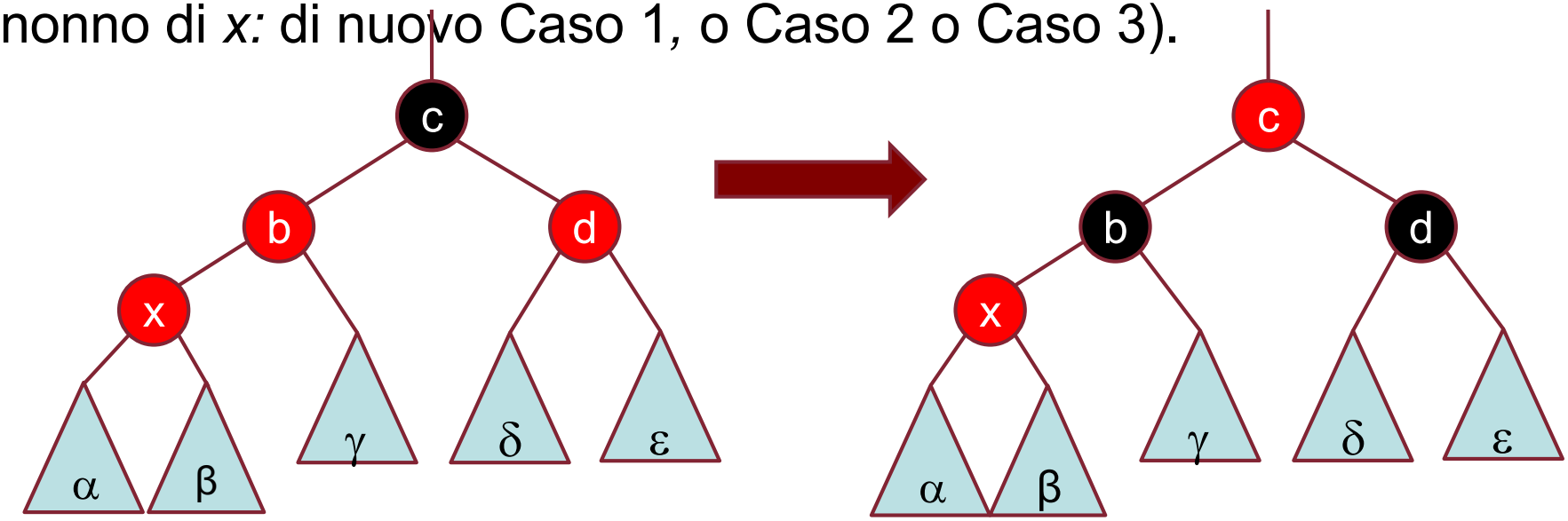
Semplicemente cambiamo il colore da rosso a nero.



Inserimento (6)

Caso 1: Il nodo *x* che crea la violazione è figlio di un nodo rosso e lo zio è rosso.

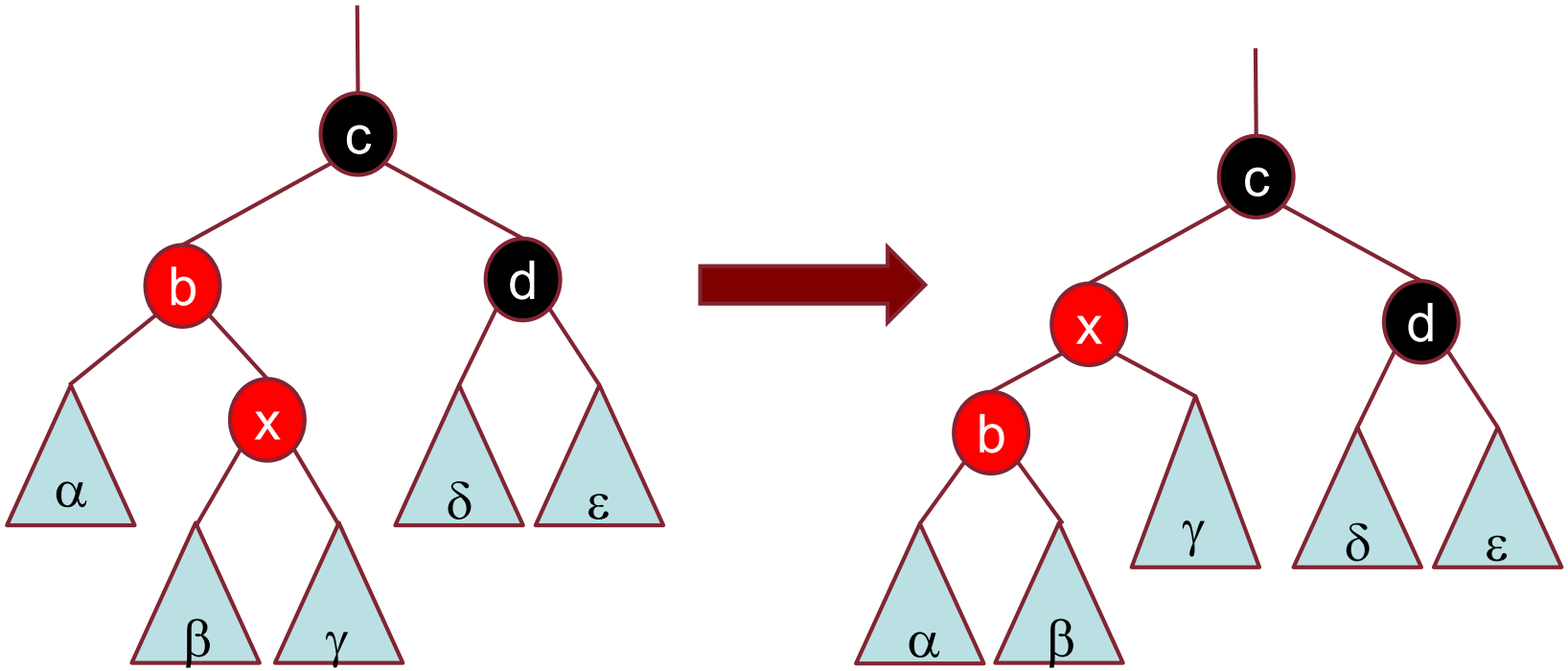
Si cambiano i colori di alcuni nodi: detto *x* il nuovo nodo che crea la violazione, si colorano di nero il padre di *x* e lo zio di *x,* di rosso il nonno di *x.* Ora, o la violazione è stata eliminata (fine inserimento), o è stata portata più in alto (a partire dal nonno di *x:* di nuovo Caso 1, o Caso 2 o Caso 3).



Inserimento (7)

Caso 2: Il nodo x che crea la violazione è figlio destro di un nodo rosso e lo zio è nero.

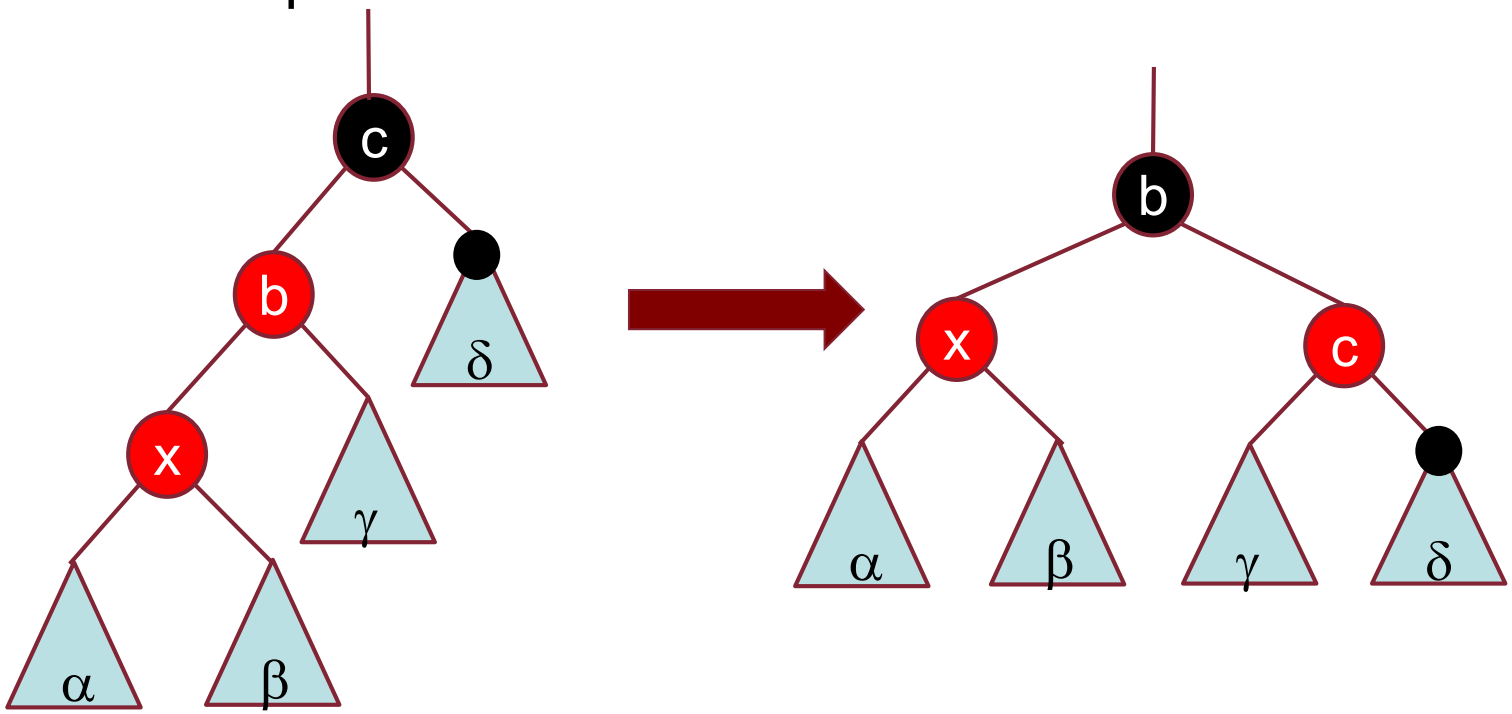
Si effettua una rotazione a sinistra imperniata sul padre di *x*, e questo conduce sempre al Caso 3.



Inserimento (8)

Caso 3: Il nodo *x* che crea la violazione è figlio sinistro di un nodo rosso e lo zio è nero.

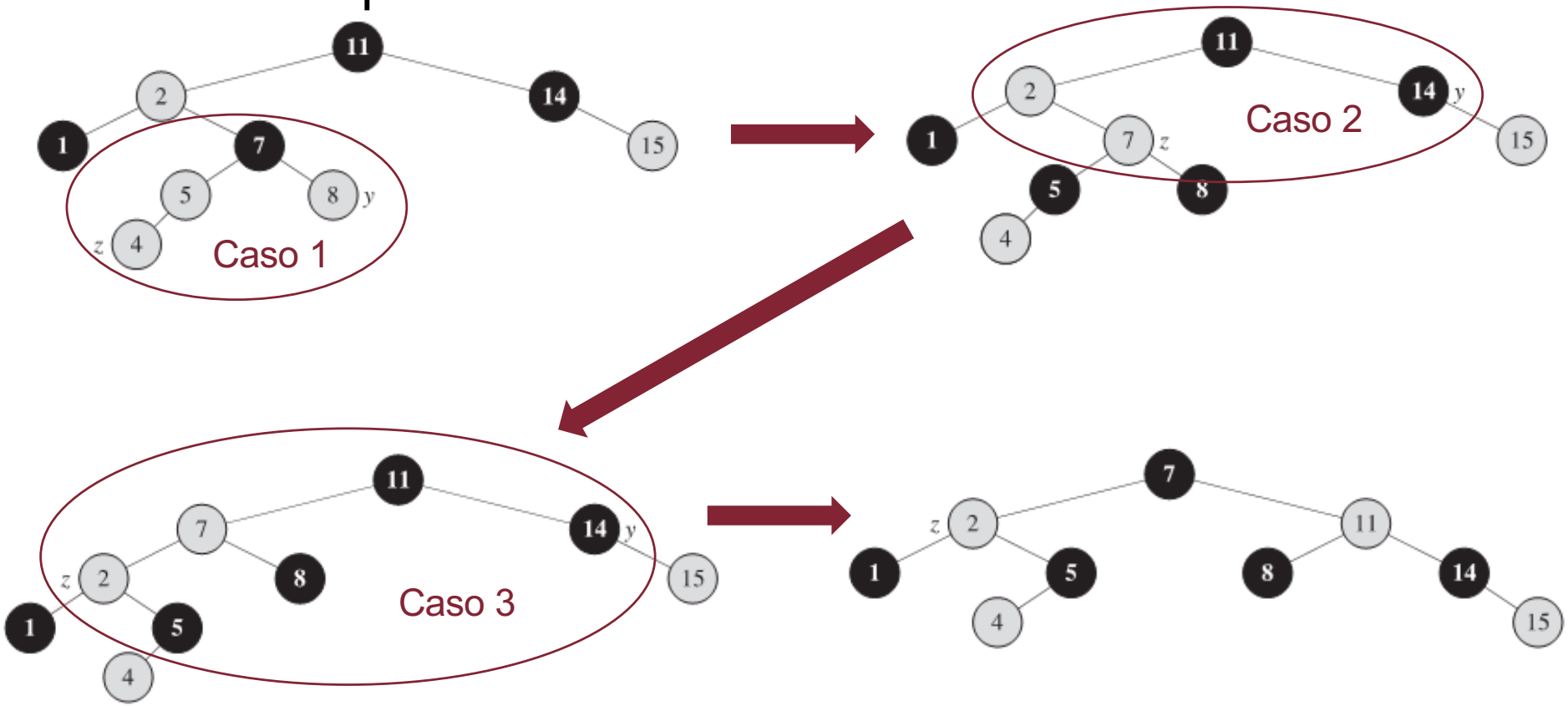
Si effettua una rotazione e si cambiano alcuni colori. Ciò garantisce sempre la soluzione della violazione.



Nota: Si procede analogamente se il padre di x è un figlio destro di suo padre.

Inserimento (9)

Un esempio:



NOTA: il problema si può propagare verso l'alto! quando la propagazione raggiunge la radice, l'altezza nera dell'albero cresce

Inserimento (10)

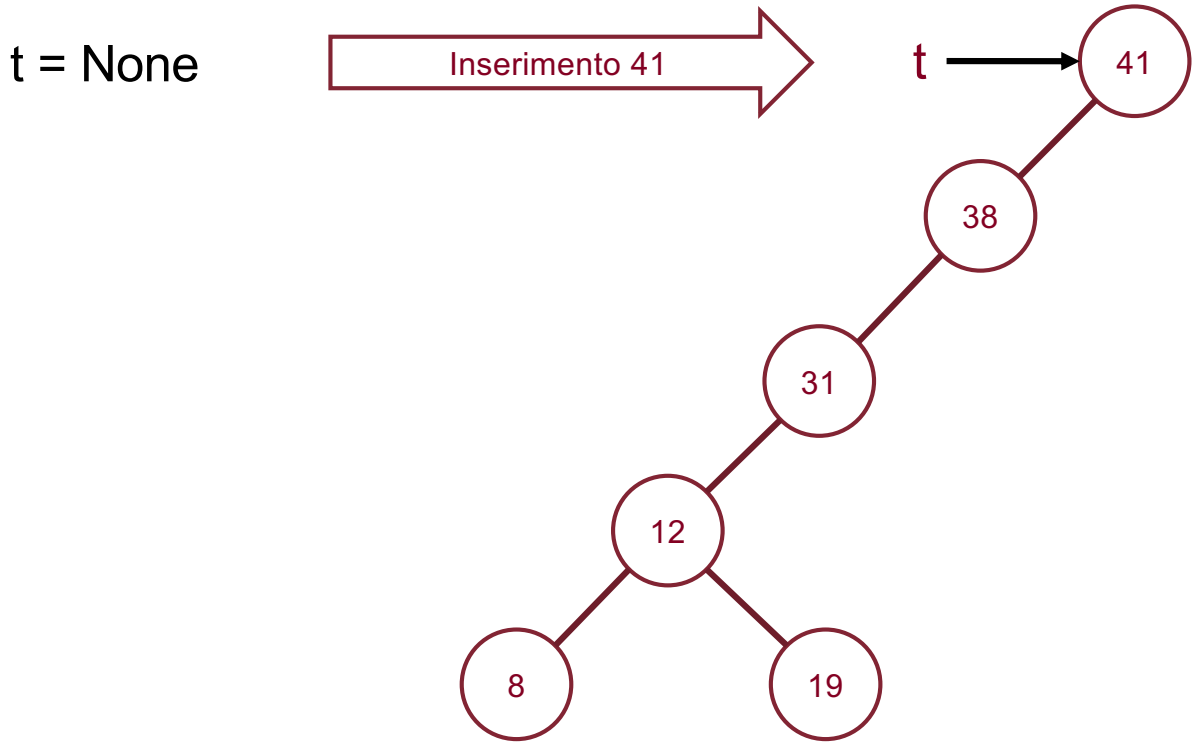
Durante un inserimento è possibile entrare ripetutamente nel caso 1 (ma sempre più in alto nell'albero), eventualmente nel caso 2 il quale si conclude sempre con il caso 3.

Poiché ogni volta che si presenta una violazione questa è più in alto nell'albero, e poiché essa è risolta sempre in tempo costante, l'inserimento in un RB albero si effettua in $O(\log n)$ tempo.

Lo stesso vale per la cancellazione, che decidiamo di omettere qui.

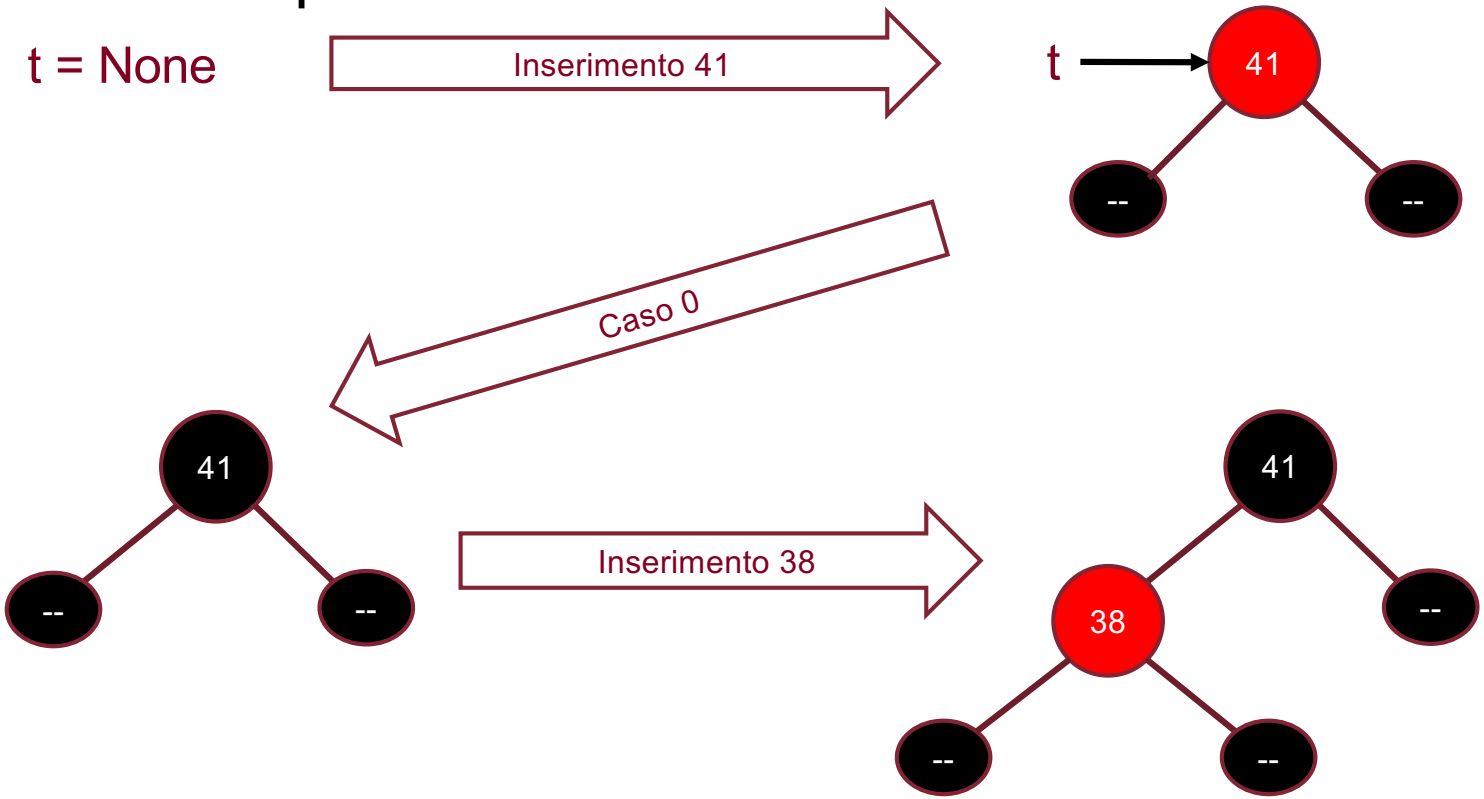
Esercizio svolto 1

Esercizio. Costruire un **ABR**, a partire dall'albero vuoto, inserendo successivamente le chiavi 41, 38, 31, 12, 19 ed 8 in quest'ordine.



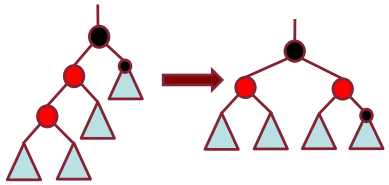
Esercizio svolto 2 (1)

Esercizio. Costruire un **RB albero**, a partire dall'albero vuoto, inserendo successivamente le chiavi 41, 38, 31, 12, 19 ed 8 in quest'ordine.



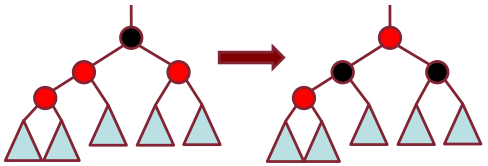
Esercizio svolto 2 (2)

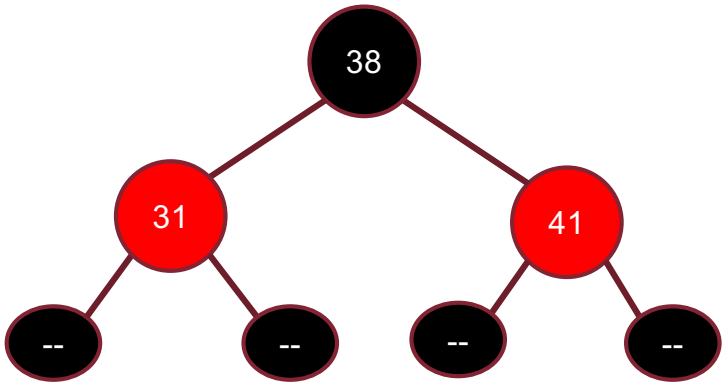
Caso 3: Il nodo che crea la violazione (31) è figlio sinistro di un nodo rosso e lo zio è nero.
Si effettua una ***rotazione*** e si cambiano alcuni colori. Ciò garantisce sempre la soluzione della violazione.



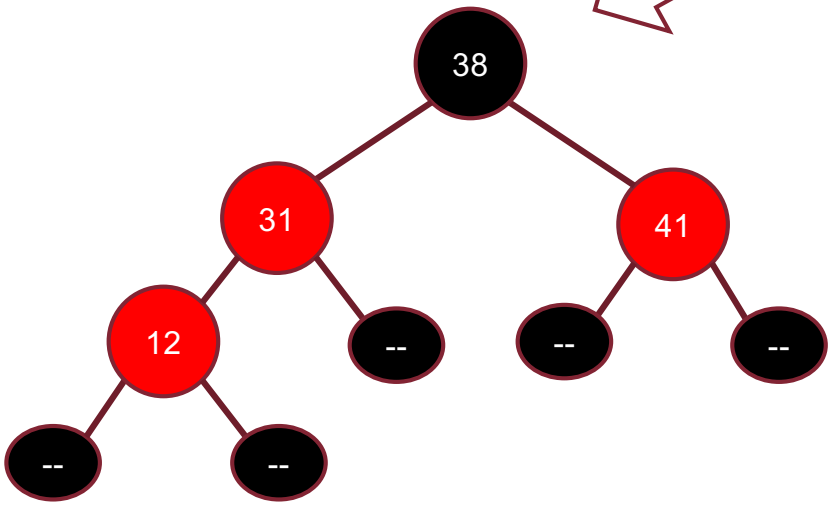
Esercizio svolto 2 (3)

Caso 1: Il nodo che crea la violazione (12) è figlio di un nodo rosso e lo zio è rosso.
Si cambiano i colori di alcuni nodi.
La violazione non è eliminata ma è stata propagata al nonno, 38.

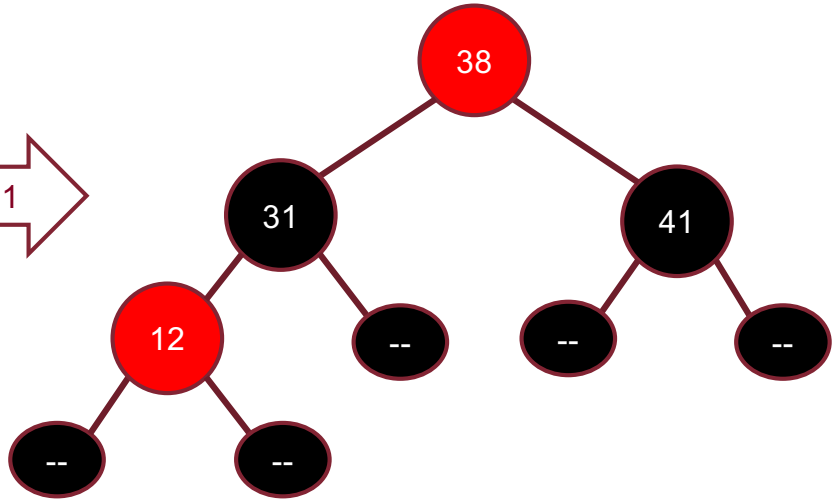




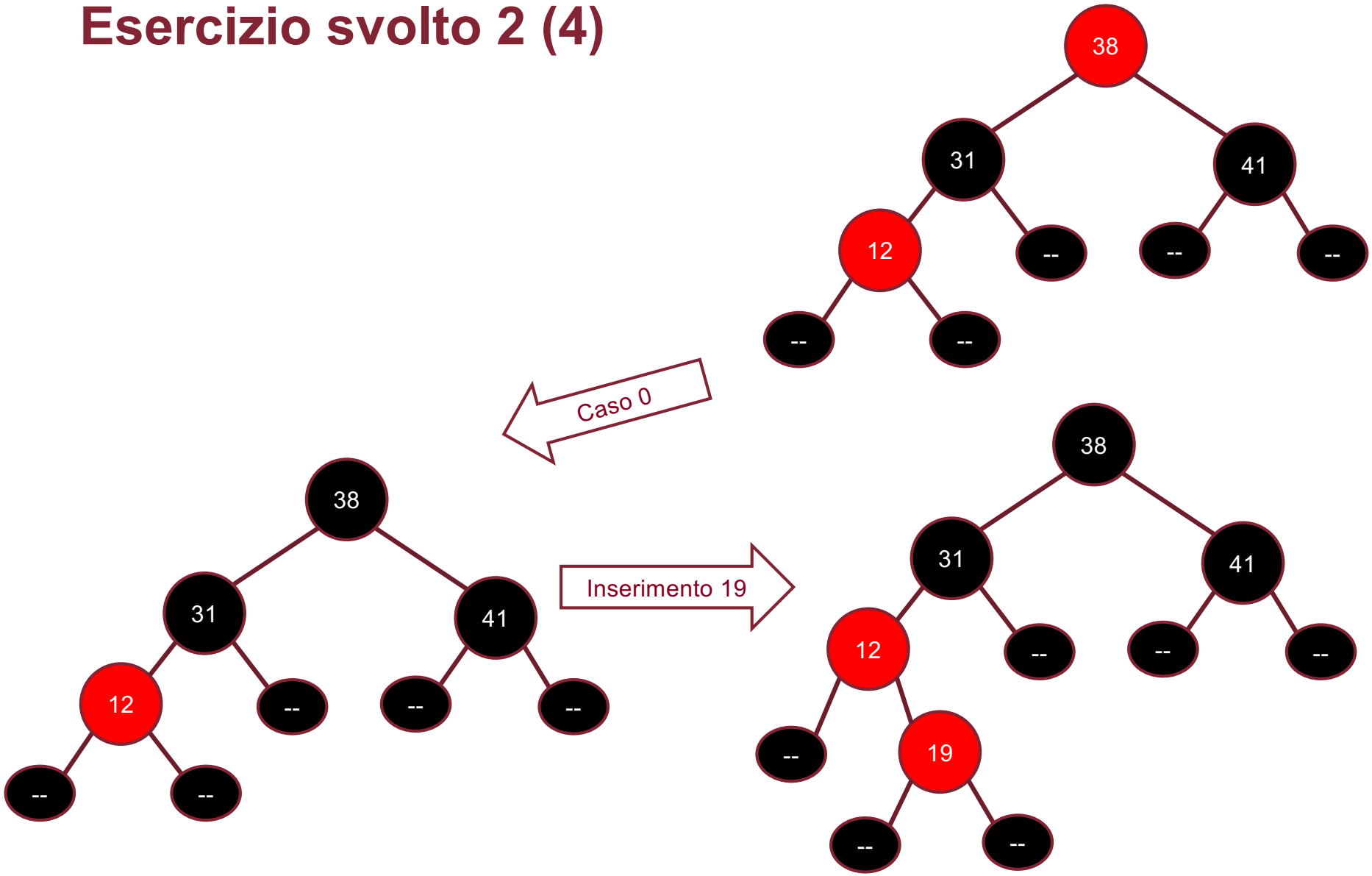
Inserimento 12



Caso 1

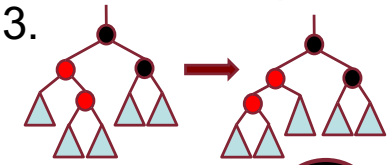


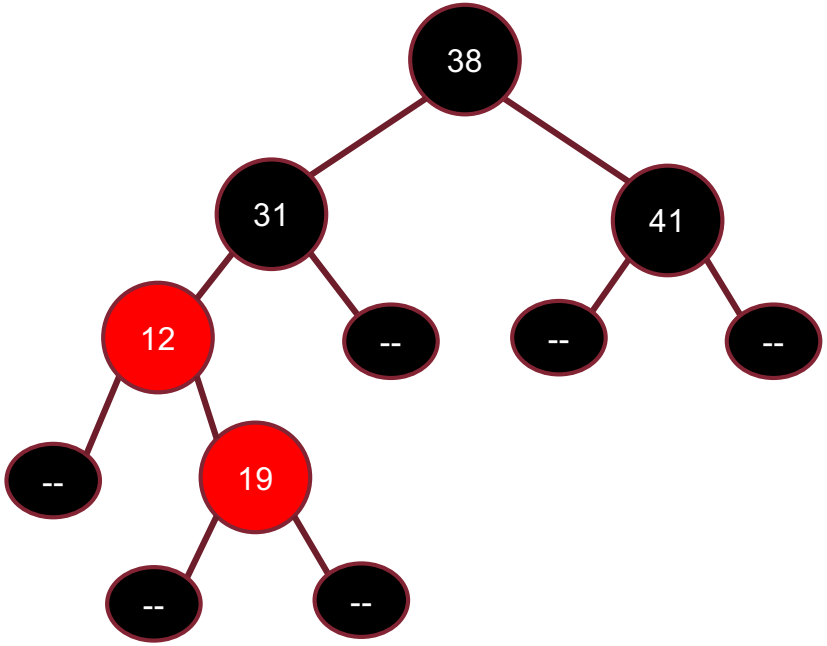
Esercizio svolto 2 (4)



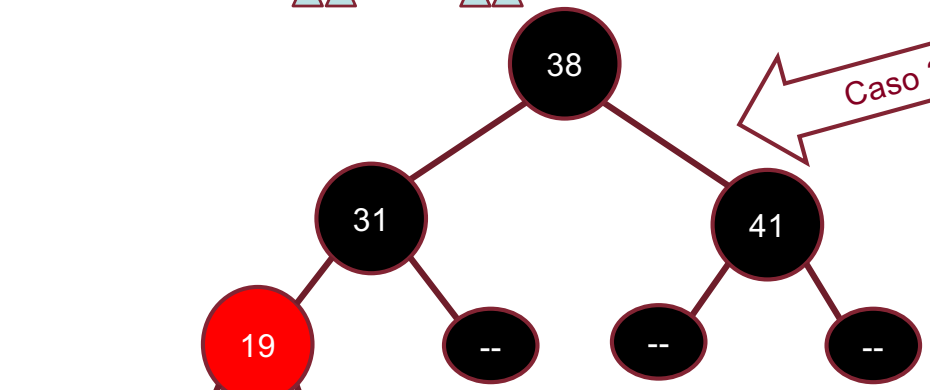
Esercizio svolto 2 (5)

Caso 2: Il nodo che crea la violazione (19) è figlio destro di un nodo rosso e lo zio è nero.
Si effettua una rotazione a sinistra imperniata sul padre di 19, e questo conduce sempre al Caso 3.

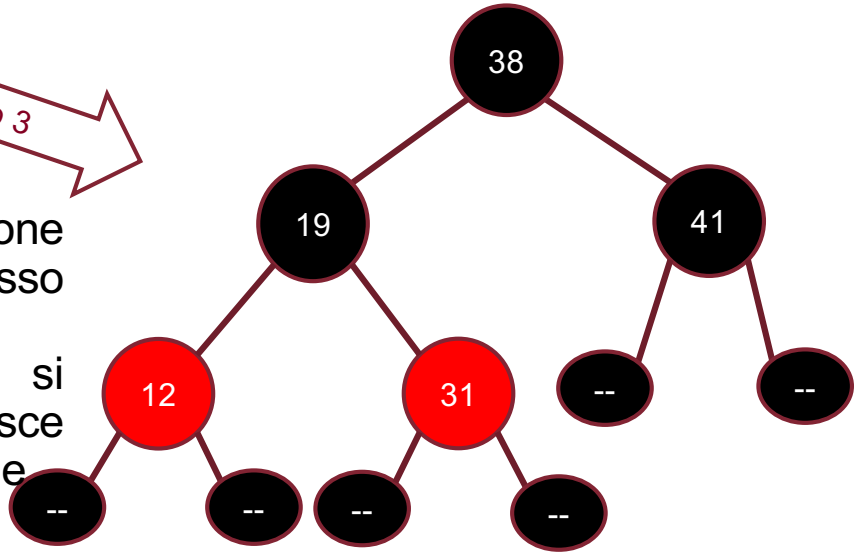




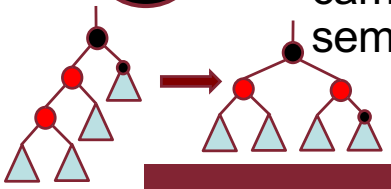
Caso 2



Caso 3

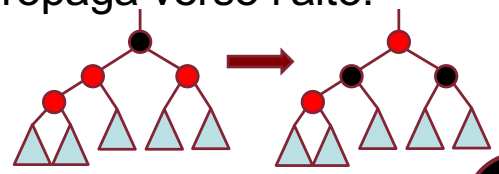


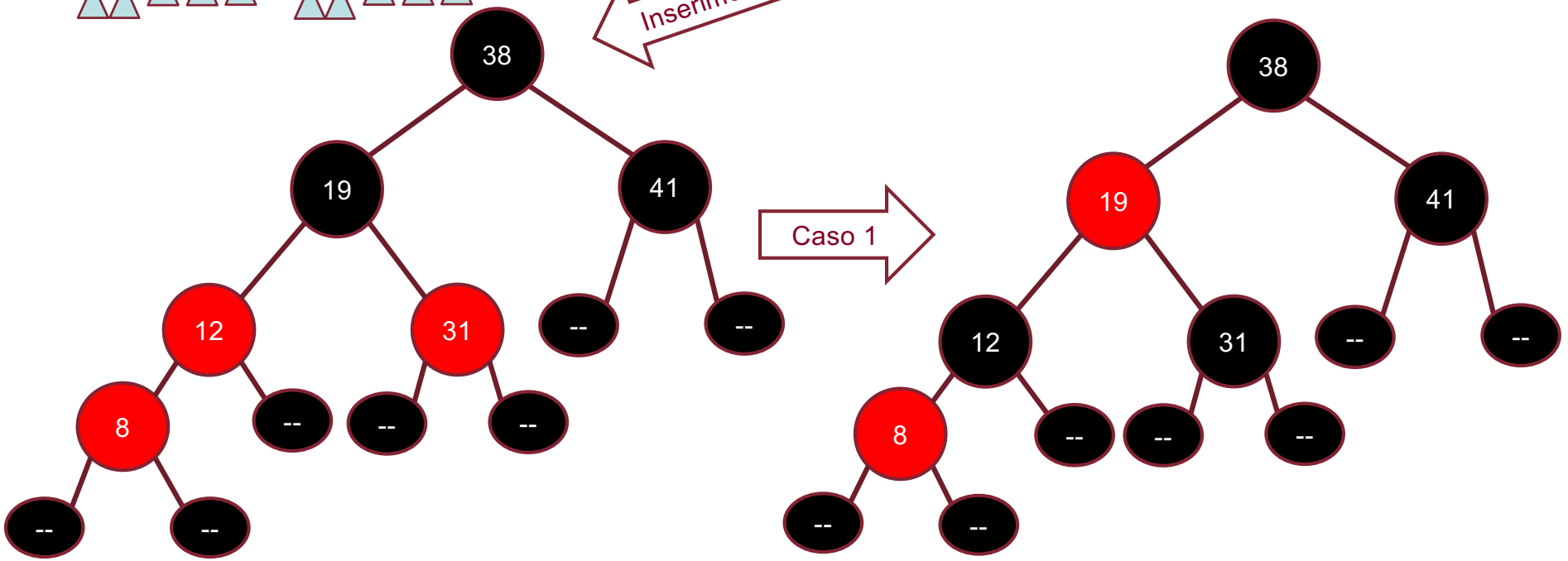
Caso 3: Il nodo che crea la violazione (12) è figlio sinistro di un nodo rosso e lo zio è nero.
Si effettua una ***rotazione*** e si cambiano alcuni colori. Ciò garantisce sempre la soluzione della violazione.



Esercizio svolto 2 (6)

Caso 1: Il nodo che crea la violazione (8) è figlio di un nodo rosso e lo zio è rosso.
Si cambiano i colori di alcuni nodi.
In questo caso la violazione non si propaga verso l'alto.





Corso di laurea in Informatica
Introduzione agli Algoritmi
Lezioni in modalità mista o a distanza

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi

- Scrivere lo pseudocodice di una funzione che ordina un array A:
  1. inserendo i suoi elementi in un ABR;
  2. ricopiando su A gli elementi incontrati eseguendo la visita inorder sull'ABR.

  Valutarne il costo computazionale. Se l'ABR è un RB albero, cambia qualcosa? se si, cosa?