Algoritmi e Strutture Dati

Alberi Rosso-Neri (RB-Trees)

Maria Rita Di Berardini, Emanuela Merelli[1]

[1]Dipartimento di Matematica e Informatica
Università di Camerino

A.A. 2007/08

Alberi Rosso-Neri: definizione

- Un albero rosso-nero (Red-Black Tree – RB tree for short) è un albero binario di ricerca in cui ad ogni nodo associamo un colore, che può essere rosso (RED) o **nero** (BLACK)

- Vincolando il modo in cui possiamo colorare i nodi lungo un qualsiasi percorso che va dalla radice ad una foglia, riusciamo a garantire che l'abero sia approssivatimente **bilanciato**

- Ogni nodo dell'albero ha quattro campi: *color*, *key*, *left*, *right* e *p*

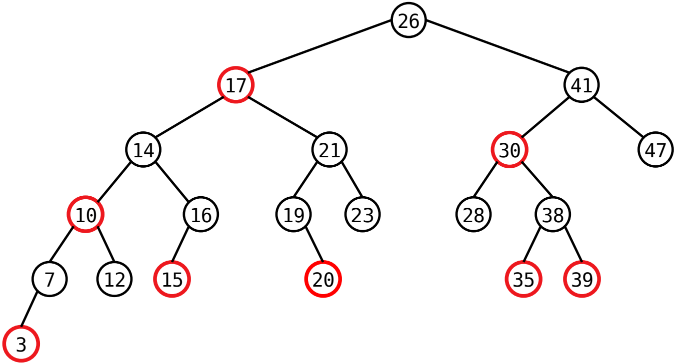
Alberi Rosso-Neri: proprietà

Un RB tree è un albero binario di ricerca che soddisfa le seguenti proprietà:

1. ogni nodo è rosso o nero
2. la radice è nera
3. ogni foglia è nera
4. se un nodo è rosso, entrambi i suoi figli devono essere neri
5. per ogni nodo n, tutti i percorsi che vanno da n alle foglie sue discendenti contengono lo stesso numero di nodi neri

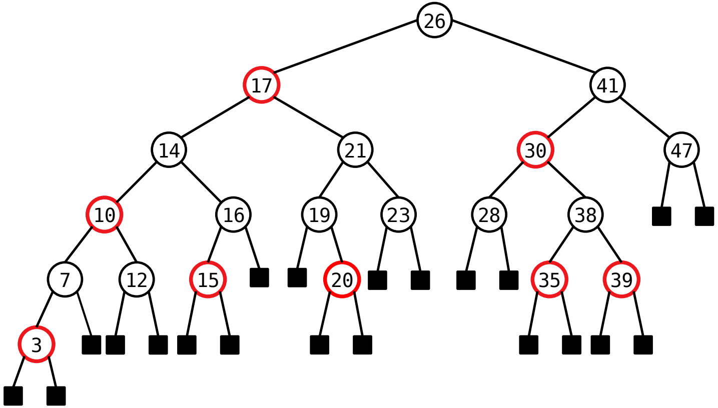
Alberi Rosso-Neri: proprietà 1

Ogni nodo è rosso o nero



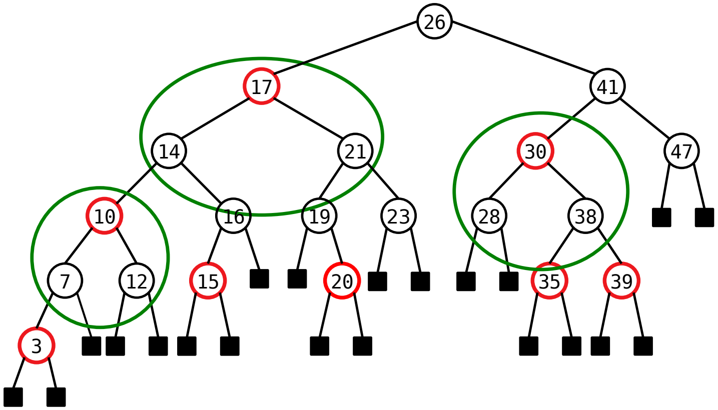
Alberi Rosso-Neri: proprietà 2

Ogni foglia è nera (basta aggiungere un ulteriore livello fittizio)



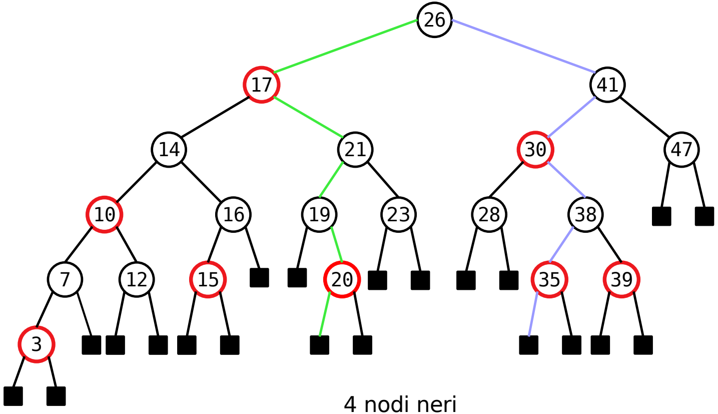
Alberi Rosso-Neri: proprietà 3

Se un nodo è rosso, entrambi i suoi figli devono essere neri



Alberi Rosso-Neri: proprietà 5

Per ogni nodo n, tutti i percorsi che vanno da n alle foglie sue discendenti contengono lo stesso numero di nodi neri



4 nodi neri

Idea di base

Per la proprietà 5 (tutti i cammini da un nodo x alle foglie hanno lo stesso numero di nodi neri) un RB tree senza nodi rossi deve essere bilanciato: tutti i suoi livelli sono completi tranne al più l'ultimo in cui può mancare qualche foglia

Non è però quasi completo perchè le foglie mancanti sull'ultimo livello non sono necessariamente quelle più a destra

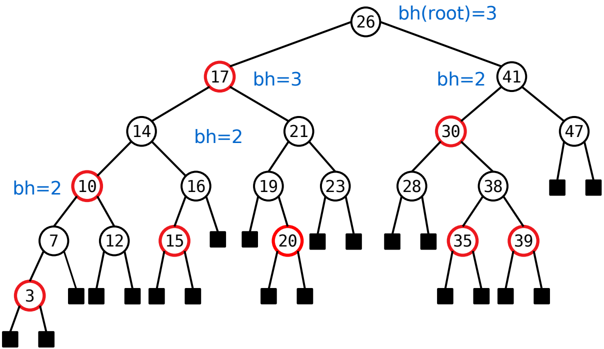
A questo albero bilanciato possiamo aggiungere "non troppi" nodi rossi (Proprietà 4 – se un nodo è rosso i suoi figli deveno essere neri)

Ciò rende l'albero "quasi bilanciato"

Black-height di un RB tree

- Se x è un nodo, definiamo $bh(x) =$ numero di nodi neri (x escluso) nel cammino da x ad una foglia
- $bh(root) =$ black-height dell'albero

Black-height di un RB tree



Nota:

- se x è rosso $bh(x)$ è uguale alla black-height del padre
- se x è nero $bh(x)$ è uguale alla black-height del padre - 1

Altezza di un RB tree

Theorem

L'altezza massima di un RB tree con n nodi interni è $2\log(n+1)$

Lemma

Il numero di nodi interni di un sottoalbero radicato in x è maggiore o uguale di $2^{bh(x)} - 1$

Altezza di un RB tree

Lemma

Il numero di nodi interni di un sottoalbero radicato in x è maggiore o uguale di $2^{bh(x)} - 1$

Per induzione sull'altezza $h(x)$ dell'albero radicato in x

Caso base: $h(x) = 0$. In questo caso x è una foglia e $bh(x) = 0$ ed il sottoalbero radicato in x ha 0 nodi interni. Inoltre

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

Passo induttivo: $h(x) > 0$. In questo caso il nodo x ha due figli: s ed d. Quale è la loro black-height? Distinguiamo due casi:

- s è rosso. Allora $bh(s) = bh(x) \geq bh(x) - 1$
- s è nero. Allora $bh(s) = bh(x) - 1 \geq bh(x) - 1$

In maniera del tutto simile abbiamo che $bh(d) \geq bh(x) - 1$

Altezza di un RB tree

Passo induttivo: $h(x) > 0$. In questo caso x ha due figli s ed d. Inoltre $bh(s) \geq bh(x) - 1$ e $bh(d) \geq bh(x) - 1$

.

Il numero dei nodi interni dell'albero radicato in x? è pari a

$$\#_{int}(x) = 1 + \#_{int}(s) + \#_{int}(d)$$

Poichè $h(s), h(r) < h(x)$ possiamo applicare l'ipotesi induttiva, che ci dice che

$$\#_{int}(s) \geq 2^{bh(s)} - 1 \geq 2^{bh(x)-1} - 1$$

e

$$\#_{int}(d) \geq 2^{bh(d)} - 1 \geq 2^{bh(x)-1} - 1$$

Allora

$$\#_{int}(x) = 1 + \#_{int}(s) + \#_{int}(d) \geq 1 + 2 \cdot (2^{bh(x)-1} - 1) = 2^{bh(x)} - 1$$

Altezza di un RB tree

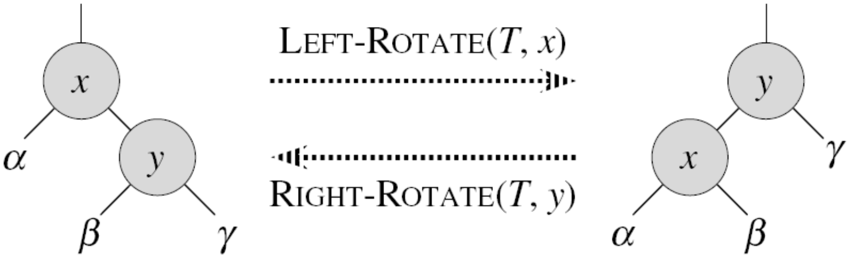
Theorem

L'altezza massima di un RB tree con n nodi interni è $2\log(n+1)$

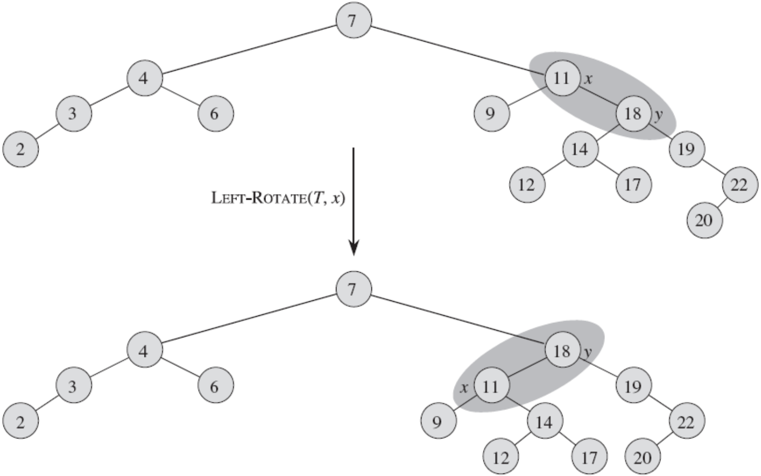
- Sia h l'altezza dell'albero
- Qualsiasi cammino dalla radice ad una foglia contiene almeno metà nodi neri e quindi almeno $h/2$ nodi neri
- $bh(T) = bh(root) \geq h/2$
- Per il lemma $n \geq 2^{bh(T)} - 1 \geq 2^{h/2} - 1$ e $n + 1 \geq 2^{h/2}$
- Quindi: $\log(n+1) \geq h/2$ e $h \leq 2\log(n+1)$

Rotazioni

Sono delle operazioni di ristrutturazione locale dell'albero che mantengono soddisfatte le proprietà dei RB trees



Rotazioni: un esempio



Operazioni su RB trees

Vediamo nel dettaglio le operazioni di **inserimento** e **cancellazione** di un nodo

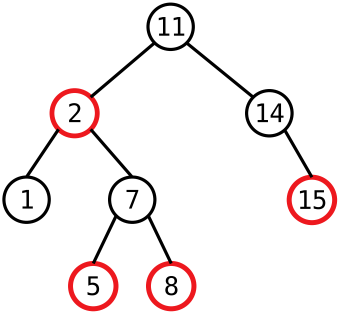
Le operazioni **Search**, **Minimum** e **Maximum**, **Successor** e **Predecessor** possono essere implementate esattamente come per gli alberi binari di ricerca "ordinari"

Inserimento

- Esattamente come per gli alberi binari di ricerca, l'inserimento di un nodo z in un RB tree cerca un cammino discendente dalla radice dell'albero fino al nodo y che diventerà suo padre
- Una volta identificato il padre y, z viene aggiunto come figlio sinistro (destro) di y se $key[z] \leq key[y]$ ($key[z] \geq key[y]$, risp.)
- Tuttavia, nel caso di RB tree abbiamo altri problemi da risolvere; innanzitutto, quale colore associamo al nodo z?
- Per la proprietà 5 (tutti i cammini da un qualsiasi nodo alle foglie sue discendenti hanno lo stesso numero di nodi neri) il colore di z deve essere rosso
- Questo ovviamente può causare la violazione di altre proprietè dei RB tree: se $color[y] = \text{RED}$ violiamo la proprietà 4
- In questo caso eseguiamo delle rotazioni + ricolazioni per ristabilire le proprietà violate

Inserimento: un esempio

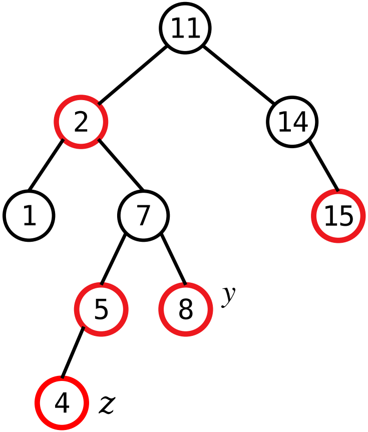
Inseriamo un nodo z con chiave 4 nell'albero



Inserimento: un esempio

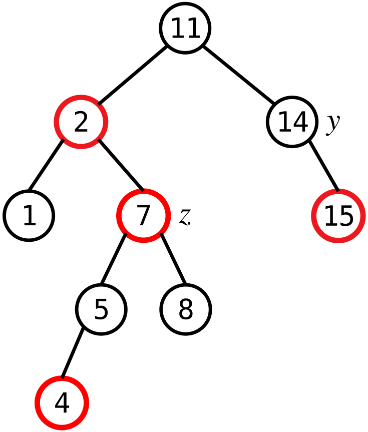
z (il nodo che causa la violazione) e suo zio y sono rossi. In questo caso:

- 5 ed 8 diventano neri
- 7 diventa rosso (per non alterare il numero di nodi neri)



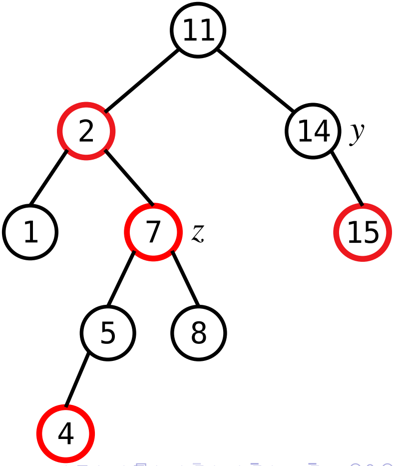
Inserimento: un esempio

z (il nodo che causa la violazione) è rosso, suo zio y è nero e z è un figlio destro. In questo caso eseguiamo una rotazione a sinistra di $p[z]$ (padre di z)

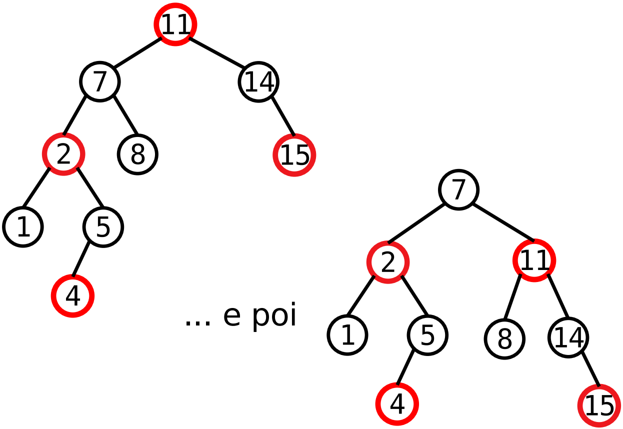


Inserimento: un esempio

z (il nodo che causa la violazione) è rosso, suo zio y è nero e z è un figlio sinistro. In questo caso 7 diventa nero, la radice 11 diventa rossa (violando la proprietà 2). Per ripristinare la proprietà 2 la root viene ruotata a destra



Inserimento: un esempio

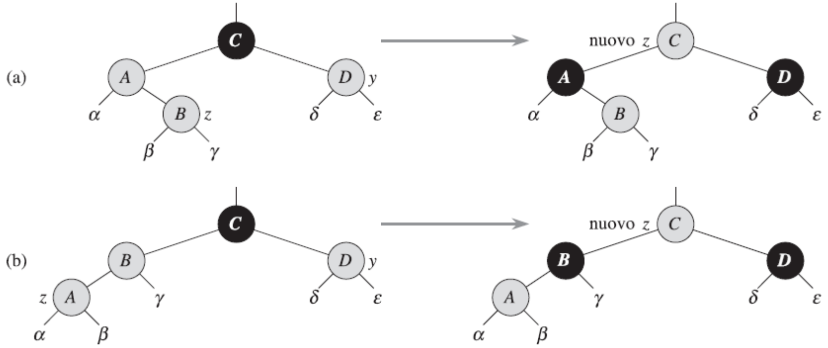


Inserimento di un nodo z

- L'inserimento di un nodo z in un RB tree cerca un cammino dalla radice dell'albero fino al nodo y che diventerà suo padre
- Una volta identificato y, z viene aggiunto come figlio sinistro o destro di y e colorato di rosso
- Eseguiamo (ricorsivamente) delle rotazioni + ricolazioni sul nodo che genera una qualche violazione delle proprietà
- Decidiamo quali rotazioni e/o ricolorazioni eseguire in base a 3 possibili casi:
  1. lo zio y di z è rosso
  2. lo zio y di z è nero e z è un figlio destro
  3. lo zio y di z è nero e z è un figlio sinistro

Caso 1: lo zio y di z è rosso

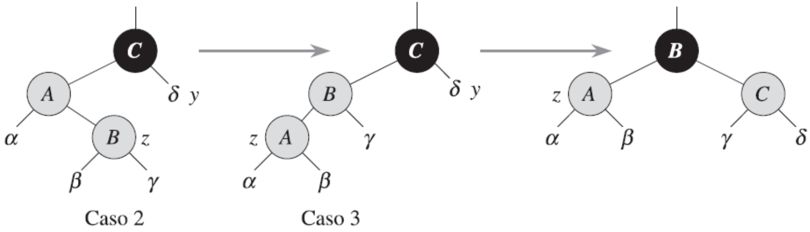
Il padre di z (A) e suo fratello y (D) – entrambi rossi – vengono colorati di nero; il $p[p[z]]$ (padre del padre di z) – nero – viene colorato di rosso e diventa il nuovo z



Casi 2 e 3: z è rosso e suo zio y è nero

Caso 2: lo zio y di z è nero e z è un figlio destro: viene ricondotto al caso 3 mediante una rotazione a sinistra di $p[z]$

Caso 3: lo zio y di z è nero e z è un figlio sinistro: scambiamo il colore di $p[z]$ (rosso) con quello di $p[p[z]]$ (nero) e ruotiamo il padre di z a destra



Cancellazione di un nodo

Assunzione: possiamo sempre assumere di eliminare un nodo che ha al massimo un figlio.

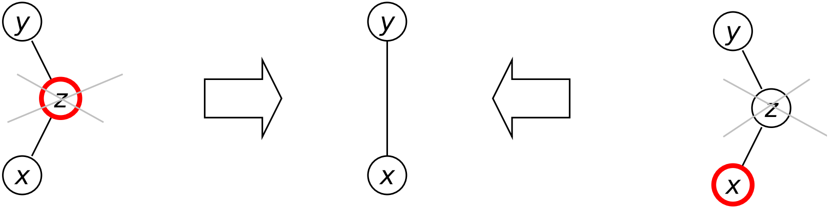
Infatti se dobbiamo cancellare un nodo z con due figli, possiamo rimpiazzare la chiave di z con quella del suo successore y, e poi rimuovere y (che ha al più solo il figlio destro) dall'albero.

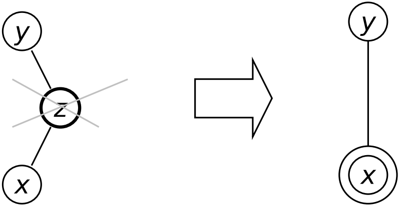
Cancellazione di un nodo con al più un figlio

Sia z il nodo da cancellare, e siano x e y il figlio ed il padre di z. Per eliminare il nodo z eseguiamo i seguenti passi

1. rimuoviamo z collegando y con x
2. se z era rosso allora y e x sono neri e terminiamo
3. se z era nero (possibile violazione della proprietà 5)
   - 3.1 se x è rosso allora coloriamo x di **nero**
   - 3.2 se x è nero allora ricoloriamo y con un colore fittizio, detto "doppio" nero, che serve a ricordarci che abbiamo collassato due nodi neri in uno violando così la proprietà 5
   - 3.3 Ripristiamo eventuali violazioni della proprietà 5

Cancellazione





x eredità il colore del padre: il doppio nero è un colore fittizio che viene usato per indicare che sia x che suo padre erano neri

Cancellazione di un nodo con al più un figlio

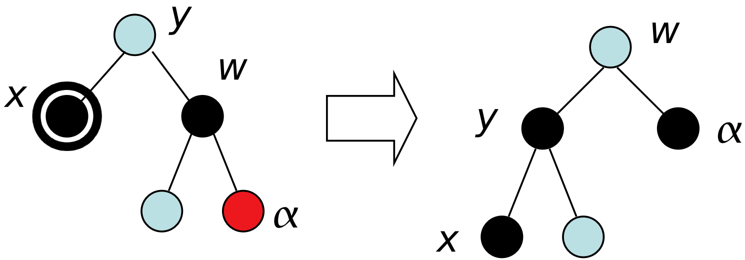
Anche in questo caso la fase di ripristino delle proprietà violate tiene conto di una serie di casi:

1. Il fratello w di x ha almeno un figlio rosso. Distinguiamo due possibili sottocasi
   1.1 il figlio rosso di w è un figlio destro
   1.2 il figlio rosso di w è un figlio sinistro
2. il fratello w di x è nero ed entrambi i suoi figli sono neri
3. il fratello w di x è rosso

Caso 1.1: fratello nero con figlio destro rosso

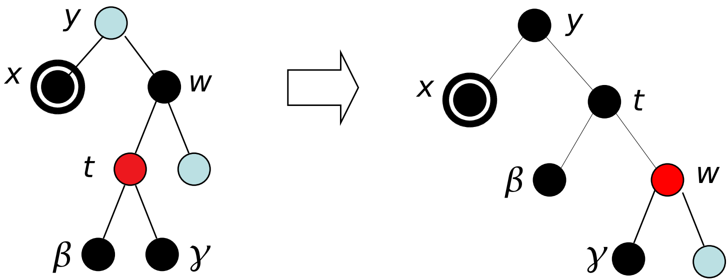
- scambiamo il colore di w (nero) con quello di y (che può essere sia nero che rosso) e ruotiamo y a sinistra
- dopo la rotazione a sinistra, il nodo y – che adesso è nero – si trova a sinistra dell'albero
- questo significa che, a causa della rotazione, nel sottoalbero sinistro viene aggiunto un nodo nero
- per ribalinciare il numero di nodi neri nel sottoalbero destro basta rendere nera la root di α
- in questo modo possiamo rimuovere il doppio nero da x rendendolo "regolarmente" nero senza violare altre proprietà

Caso 1.1: fratello nero con figlio destro rosso



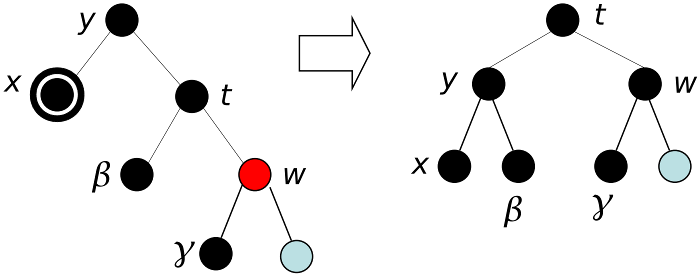
Caso 1.2: fratello nero con figlio sinistro rosso

Può essere ricondotto al caso 1.1 scambiando il colore di w con quello della root t del suo figlio sinistro (che sappiamo essere rossa) e ruotando t a destra



Caso 1.2: fratello nero con figlio sinistro rosso

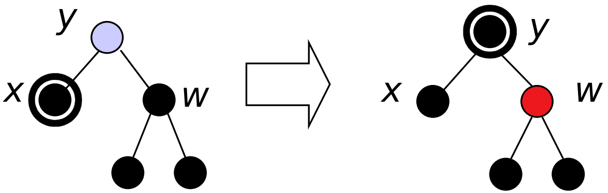
A questo punto possiamo eseguire le ricolorazioni/rotazioni descritte per il caso 1.1 (notare che in questo caso α è il sottoalbero la cui root è w) e rimuovere così il doppio nero.



Caso 2: il fratello nero con entrambi i figli neri

Poichè anche w è nero, togliamo un nero sia da x che da w lasciando x con un solo nero e w rosso.

Per compensare la rimozione di un nero sia da x che da w coloriamo il y (che originariamente era rosso oppure nero) con un doppio nero. A questo punto y diventa il nuovo x



Caso 3: il fratello w di x è rosso

Poichè w è rosso, suo padre y ed i suoi figli (le root degli alberi α e β) devono essere neri

Possiamo scambiare il colore di w con quello di y e ruotare y a sinistra senza violare nessuna delle proprietà red-black

A questo punto x è sceso di un livello a sinistra ed ha un nuovo fratello (la root di α) che è nero; abbiamo trasformato questo caso in uno dei casi precedenti