

Albero R-N = albero binario di ricerca aumentato

→ vertici colorati in modo che:

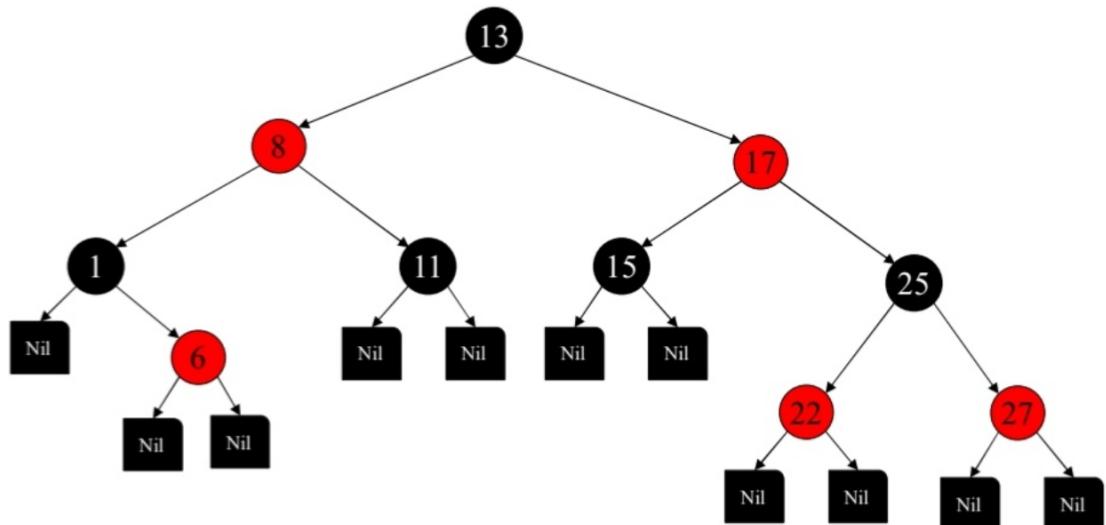
- REGOLA del NERO → radice e sottoalberi vuoti sono neri
- REGOLA del ROSSO → se un nodo è rosso, tutti i figli sono neri
- REGOLA del CAMMINO → per ogni nodo x , tutti i cammini da x ad una foglia hanno lo stesso numero di nodi neri

foglia = nil

$bh(x)$ = altezza nera di x

= numero di nodi neri su un ramo x -foglia (x escluso)

esempio:



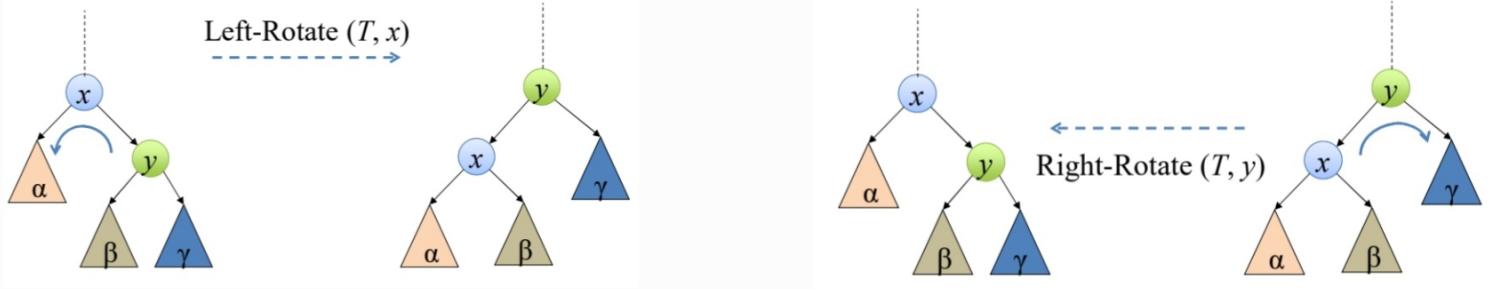
PROPRIETÀ

L'altezza massima di un albero R-N con n nodi è $2 \log_2(n+1)$

⇒ Ricerca, inserimento e cancellazione:

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

ROTAZIONI



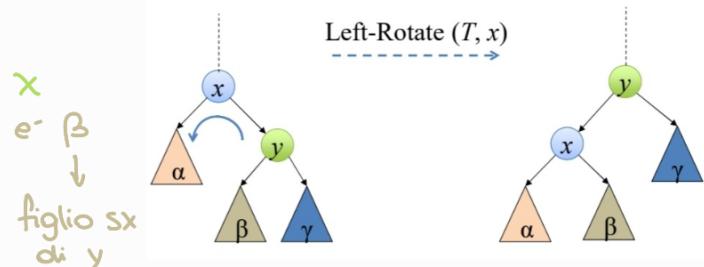
- Dopo una rotazione, l'albero rimane di ricerca
- Potrebbe abbassare l'altezza dell'albero

LEFT-ROTATE(T, x)

$y \leftarrow x.right$ riferimento al figlio dx di x
 $x.right \leftarrow y.left$ → il nuovo figlio dx di x è β
 if $y.left \neq nil$ then $y.left.parent \leftarrow x$
 end if
 $y.parent \leftarrow x.parent$
 if $x.parent = nil$ then $T \leftarrow y$ se x era radice → y nuova radice
 else

if $x = x.parent.left$ then $x.parent.left \leftarrow y$
 else $x.parent.right \leftarrow y$
 end if

end if
 $y.left \leftarrow x$ x = nuovo figlio sx di y
 $x.parent \leftarrow y$ y = nuovo padre di x



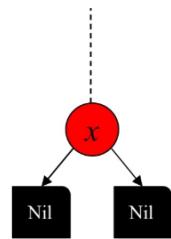
→ RIGHT-ROTATE e' speculare

INSERIMENTO

Avviene in due fasi :

1) Inserimento di x **ROSSO** come per gli alberi di ricerca

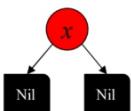
2) Ripristino delle proprietà R-N con rotazioni e ricolorazioni



Siano

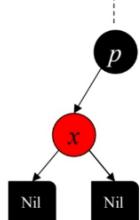
- x : nuovo nodo
- p : x .parent
- g : p .parent (nonno di x)
- u : fratello di p

► Se l'inserimento è in radice \rightarrow cambio colore in nero



Altrimenti:

► Caso 0 : p è nero

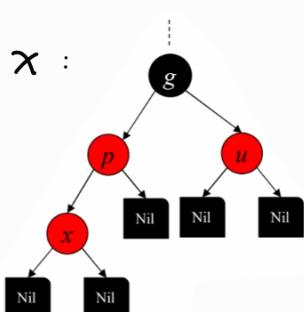


$\rightarrow \text{bh}(p)$ non cambia

\rightarrow nessuna regola violata

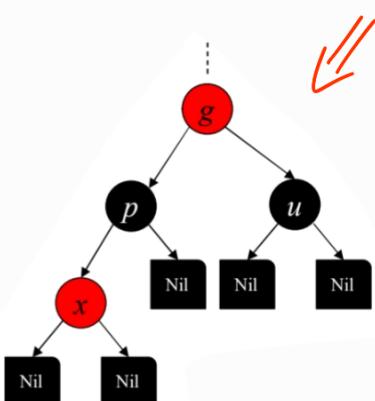
► Caso 1 : All'inizio u e p sono rossi

Inserisco x :



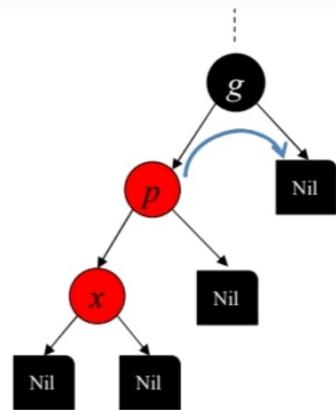
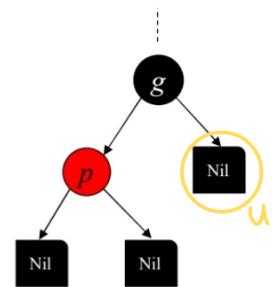
x deve essere ROSSO
perché, se fosse nero, avremmo due neri consecutivi nel ramo \rightarrow viola regole del cammino

A questo punto, p e u devono essere NERI, ma c'è un nero di troppo nel cammino di $g \Rightarrow g$ diventa **ROSSO**
(x nero che g non sia radice)

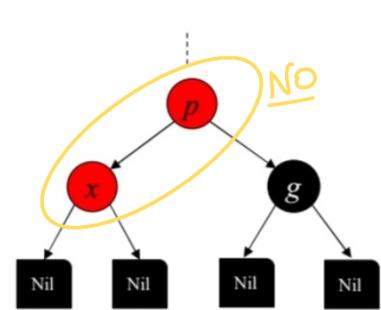


ATTENZIONE: se il padre di g fosse rosso,
si violerebbe le regole del rosso
(lo vedremo)

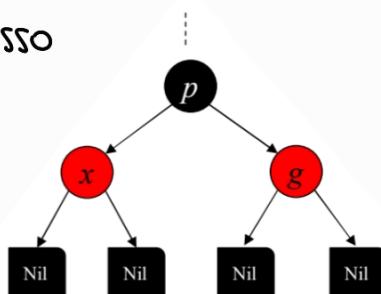
► CASO 2: All'inizio u è Nil (\Rightarrow nero) e p rosso
(x sarà figlio sx)



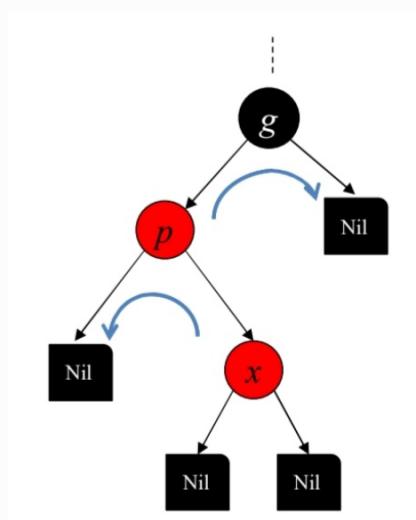
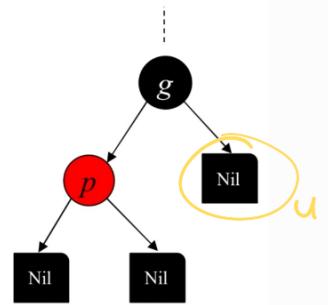
se rendo p nero, non rispetto le regole del cammino \Rightarrow RUOTO



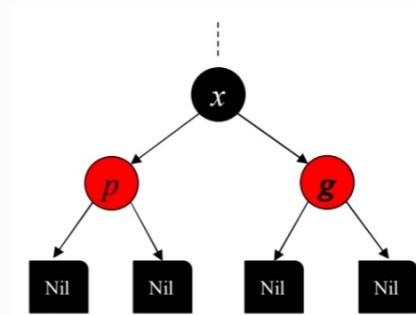
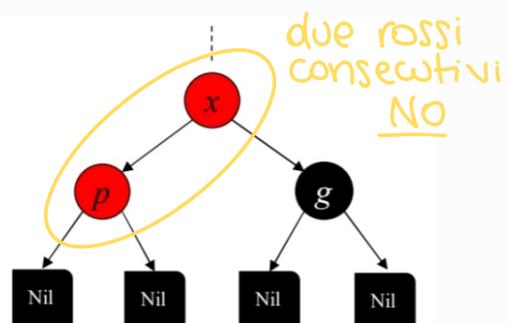
Ora, per rispettare le regole, rendo p nero e g rosso



► Caso 3: All'inizio u e' nero (e nil)
e x sera' figlio dx



→ RUOTO, ottenendo
x in radice con
p figlio sx e g figlio dx



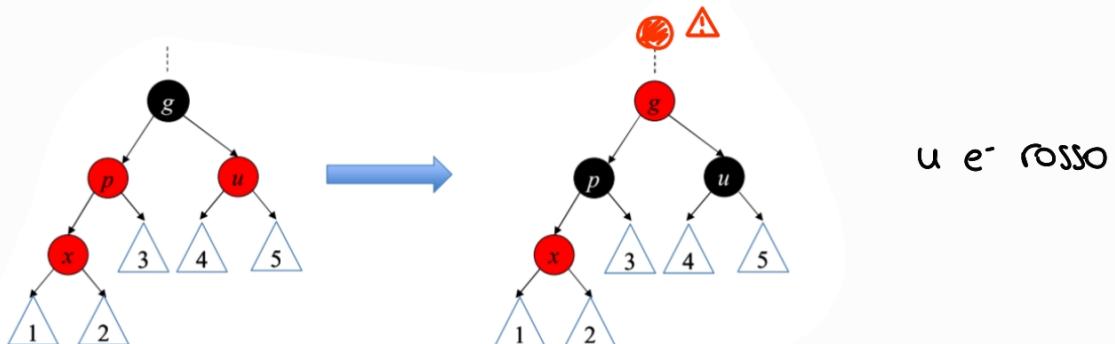
Ok

Poi, per ristabilire le
proprietà R-N, rendo x nero e
g rosso.

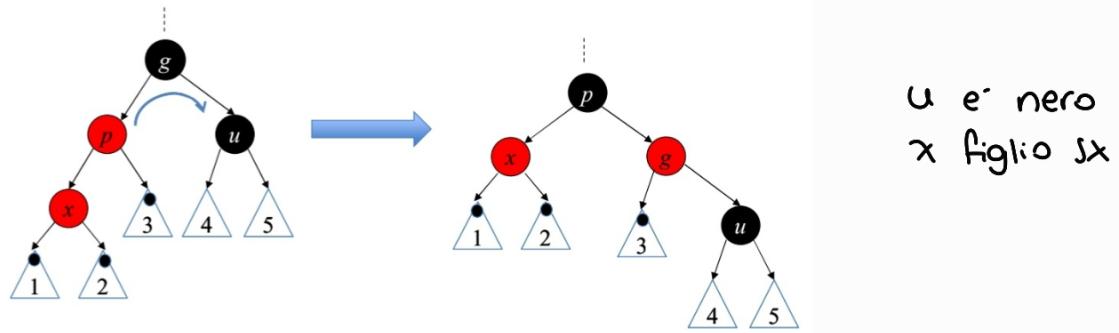
Come abbiamo visto, nel CASO 1 puo' capitare che le regole del rosso venga violata se il padre di g e' rosso.

In tal caso, dobbiamo risolvere nuovamente i 3 casi considerando che i nil non siano vuoti.

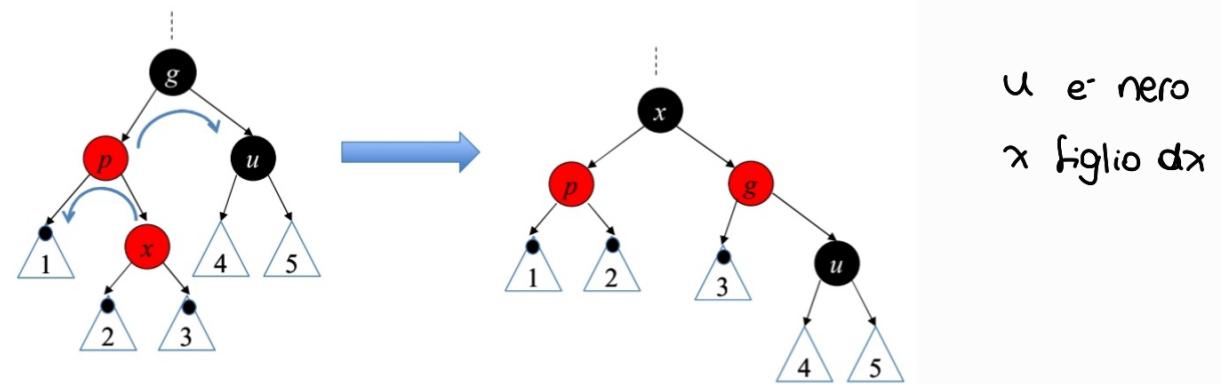
CASO 1 :



CASO 2 :



CASO 3 :



IN PSEUDOCODE :

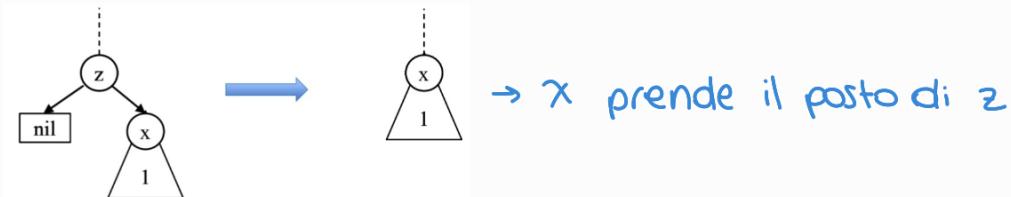
```
RB-INSERT-FIXUP( $T, x$ )      ▷  $x$  è il nodo che può creare problemi
  while  $x.p.color = red$  do
    if  $x.p = x.p.p.left$  then          ▷ padre di  $x$  è figlio sinistro?
       $u \leftarrow x.p.p.right$            ▷  $u$  è lo zio di  $x$ 
      if  $u.color = red$  then          ▷ caso 1?
         $x.p.color \leftarrow black$        ▷ caso 1
         $u.color \leftarrow black$          ▷ caso 1
         $x.p.p.color \leftarrow red$      ▷ caso 1
         $x \leftarrow x.p.p$              ▷ caso 1
      else
        if  $x = x.p.right$  then      ▷ caso 3?
           $x \leftarrow x.p$                ▷ caso 3
          LEFT-ROTATE( $T, x$ )          ▷ caso 3
           $x.p.color \leftarrow black$      ▷ caso 2 e 3
           $x.p.p.color \leftarrow red$     ▷ caso 2 e 3
          RIGHT-ROTATE( $T, x.p.p$ )     ▷ caso 2 e 3
      else
        {tutto il corpo del if esterno con
         left e right scambiati}      ▷ padre di  $x$  è figlio destro
     $T.root.color \leftarrow black$ 
```

CANCELLAZIONE

Avviene in due fasi:

- 1) Cancellazione come in un albero di ricerca ordinario
- 2) Ripristino delle regole con ricolorazioni / rotazioni

CASO A: il nodo da cancellare ha solo figlio dx



CASO B: il nodo da cancellare ha solo figlio Sx

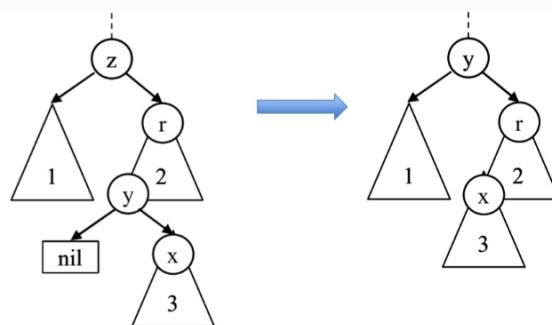


CASO C: il nodo da cancellare ha due figli e il suo successore non e' suo figlio

y = successore di z

y prende il posto di z

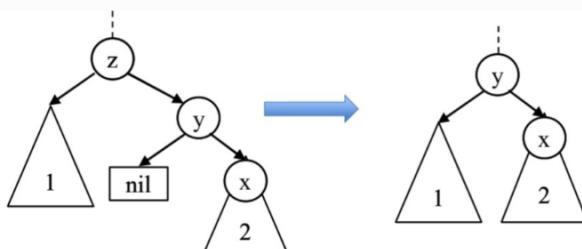
e x (figlio di y) prende il posto di y



CASO D: il nodo da cancellare ha due figli e il suo successore e' suo figlio

y prende il posto di z

e x (figlio di y) prende il posto di y



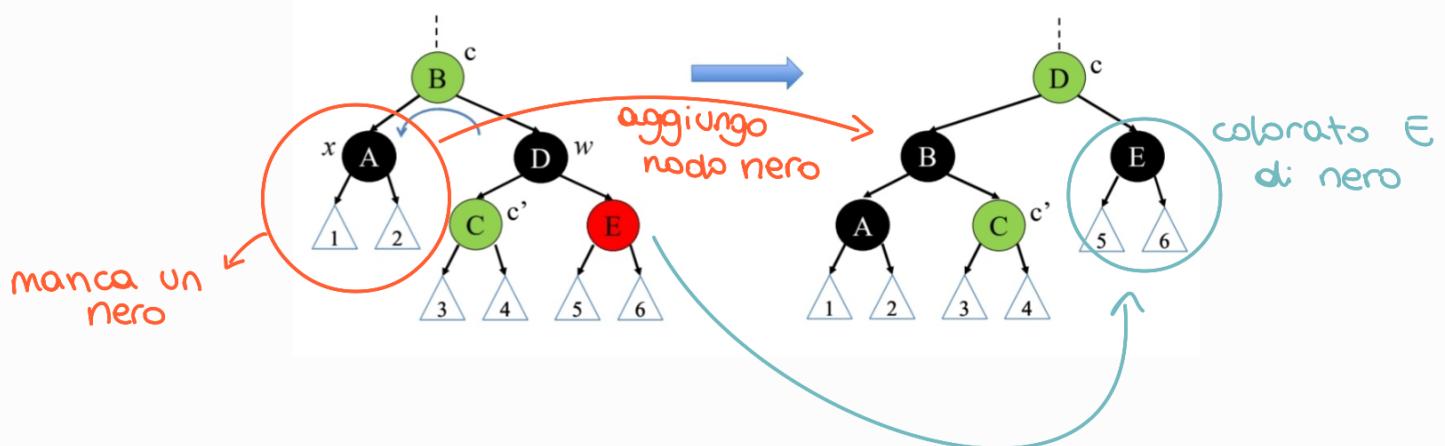
In tutti i casi :

- x mantiene il suo colore
- y prende il colore di z
- serve variabile `lost-color` per memorizzare il colore del nodo perso

FIX-UP ←

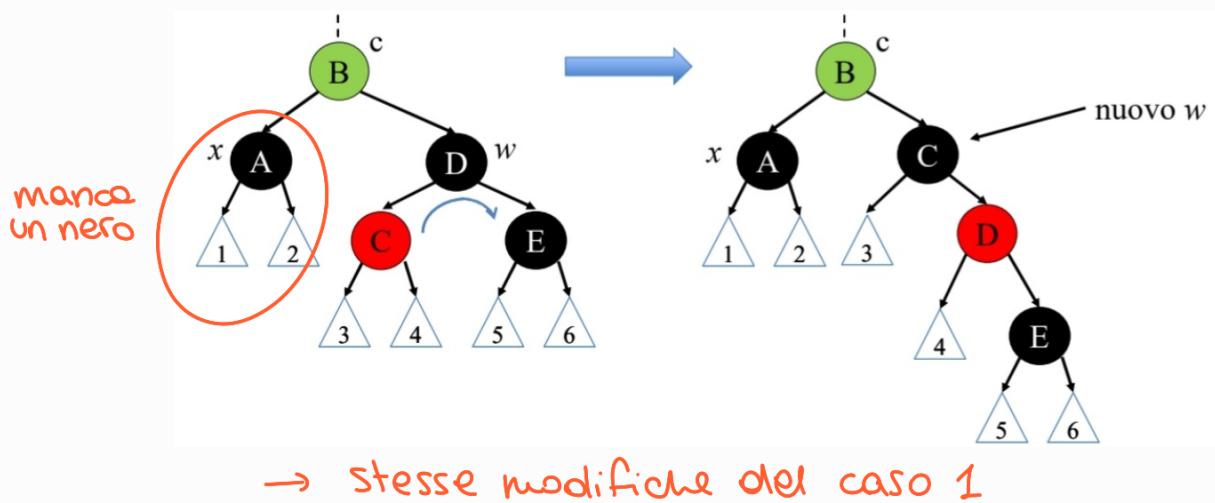
Analisi casistiche in cui *lost-color* e' nero e nodo x e' nero:

CASO 1: w nero e figlio dx di w e' rosso

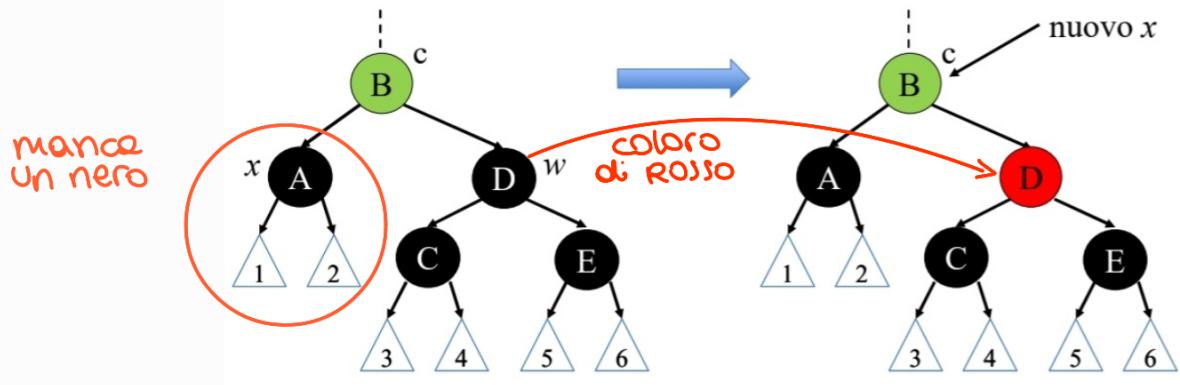


Se D e' la radice ed e' rosso \rightarrow OK

CASO 2: w e' nero , il suo figlio sx e' rosso e
il suo figlio dx e' nero



CASO 3: w e' nero e i suoi figli sono neri

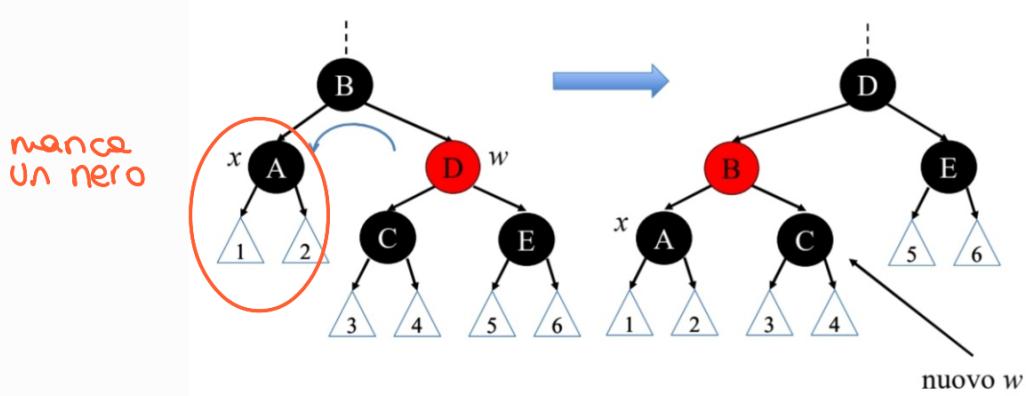


Ora mance un nero da B in su !

Se B rosso \rightarrow coloro di nero \rightarrow Ok

Se B nero \rightarrow ricomincia dal livello di B
e B e' il nuovo x

CASO 4 : w e' rosso (\Rightarrow figli neri)



\rightarrow le modifiche portano ad uno dei 3 casi precedenti
perche il fratello di x ora e' nero.

PSEUDOCODE

```
RB-DELETE-FIXUP( $T, x$ )           ▷  $x$  è il nodo che può creare problemi
  while  $x \neq T.root \wedge x.color = black$  do
    if  $x = x.p.left$  then          ▷  $x$  è figlio sinistro?
       $w \leftarrow x.p.right$           ▷  $w$  è il fratello di  $x$ 
      if  $w.color = red$  then          ▷ caso 4?
         $w.color \leftarrow black; x.p.color \leftarrow red$           ▷ caso 4
        LEFT-ROTATE( $T, x.p$ );  $w \leftarrow x.p.right$           ▷ caso 4
      if  $w.left.color = black \wedge w.right.color = black$  then          ▷ caso 3?
         $w.color \leftarrow red; x \leftarrow x.p$           ▷ caso 3
      else
        if  $w.right.color = black$  then          ▷ caso 2?
           $w.left.color \leftarrow black; w.color \leftarrow red$           ▷ caso 2
          RIGHT-ROTATE( $T, w$ );  $w \leftarrow x.p.right$           ▷ caso 2
         $w.color \leftarrow x.p.color; x.p.color \leftarrow black$           ▷ caso 1
         $w.right.color \leftarrow black; LEFT-ROTATE( $T, x.p$ )$           ▷ caso 1
         $x \leftarrow T.root$           ▷ caso 1
    else
      {tutto il corpo del if esterno con left e right scambiati}
     $x.color \leftarrow black$ 
```