Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

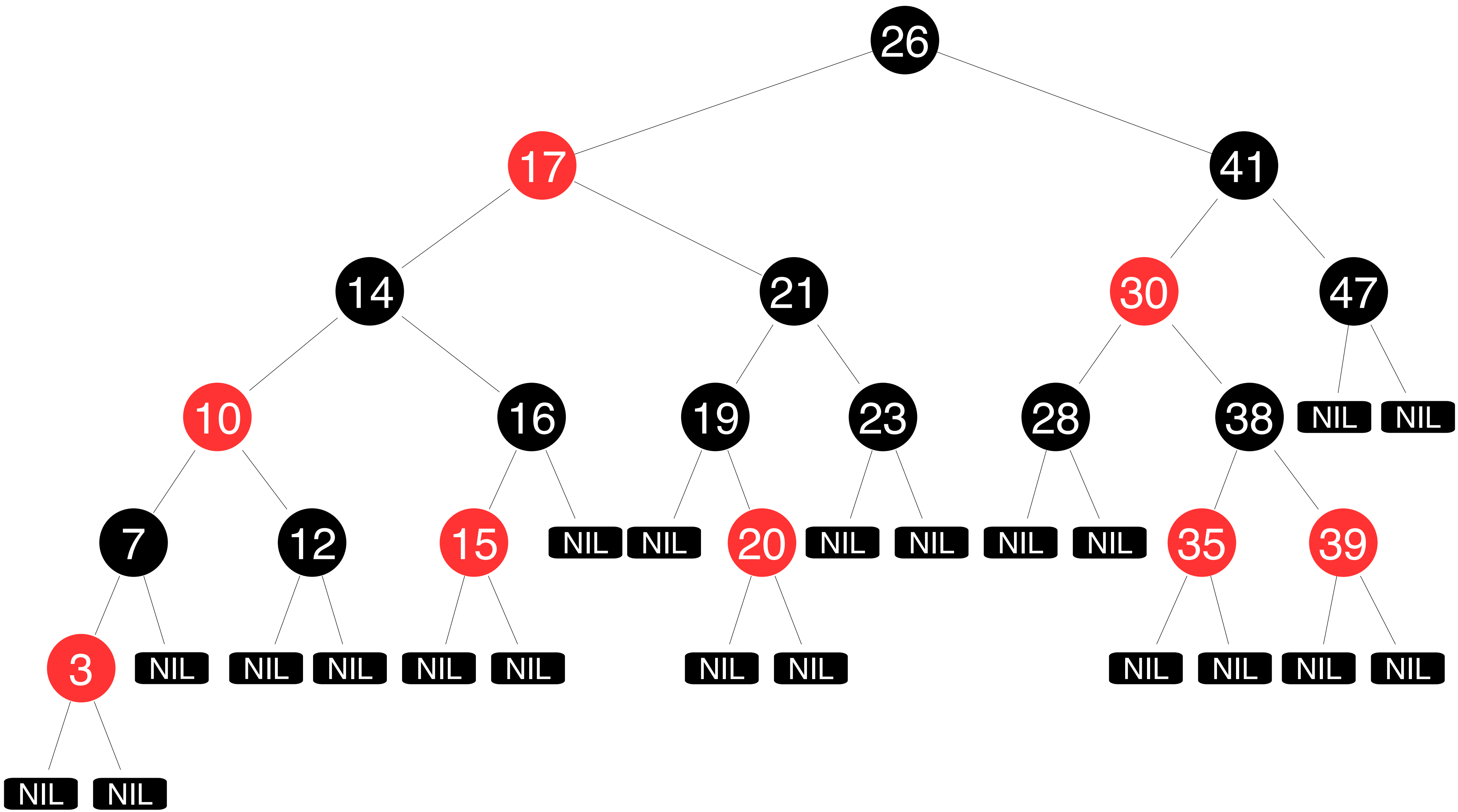
Corso di Algoritmi e Strutture Dati

Alberi rosso-neri

Alberi rosso-neri

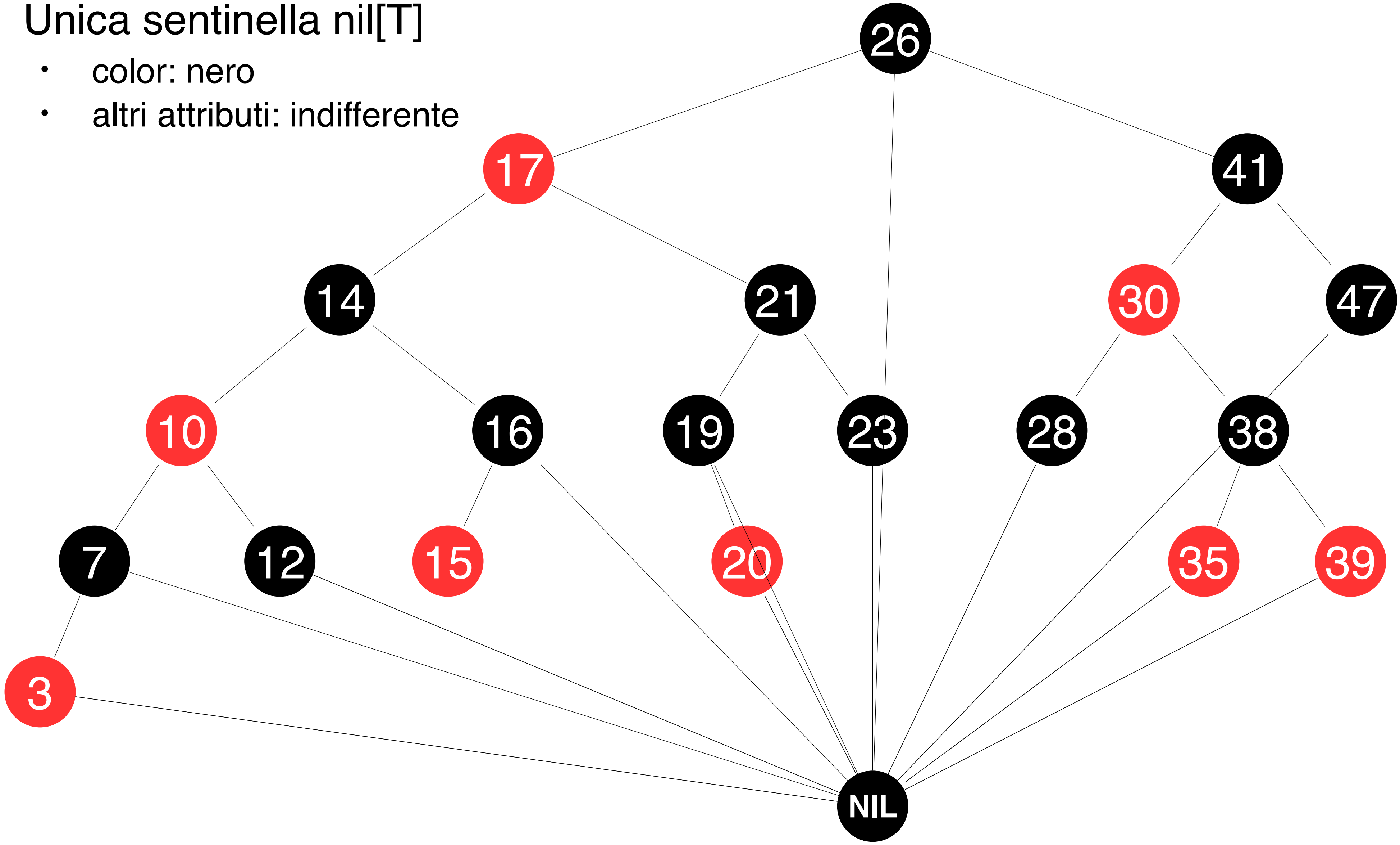
- Gli alberi rosso-neri sono alberi di ricerca in cui ogni nodo ha un attributo addizionale color, che può valere rosso o nero
- I puntatori a NIL sono sostituiti da nodi foglia esterni
- Devono essere soddisfatte le seguenti proprietà
    1. Ogni nodo o è rosso o è nero
    2. La radice è nera
    3. Ogni nodo foglia esterno (NIL) è nero
    4. Se un nodo è rosso, entrambi i figli sono neri
    5. Per ogni nodo, tutti i percorsi dal nodo alle foglie contengono lo stesso numero di nodi neri
- <u>Gli alberi rosso-neri garantiscono che nessun percorso dalla radice ad una foglia è lungo più del doppio di un altro percorso</u>

Alberi rosso-neri



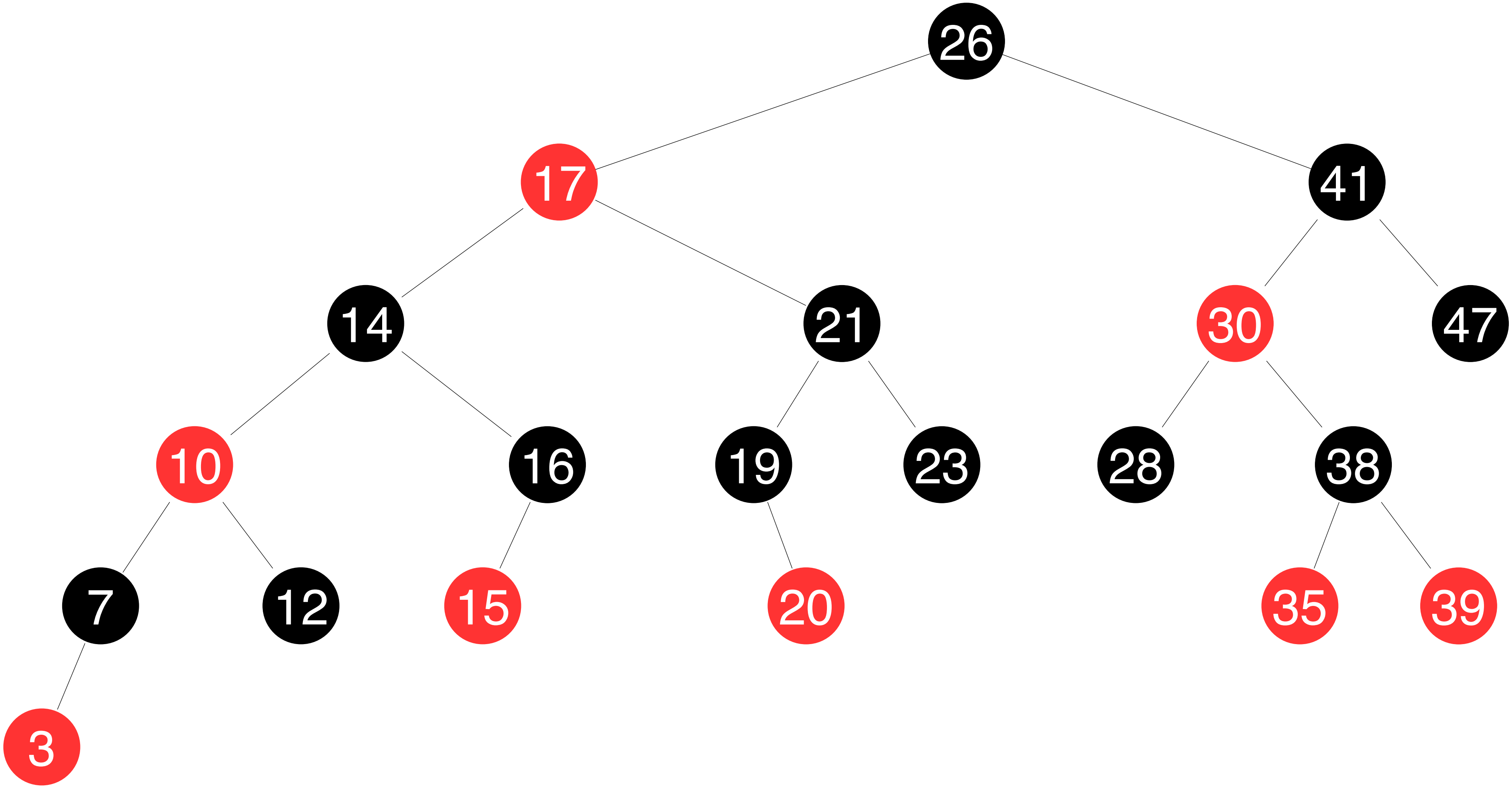
Alberi rosso-neri

- Unica sentinella nil[T]
  - color: nero
  - altri attributi: indifferente



Alberi rosso-neri

Per comodità di visualizzazione, omettiamo la sentinella



Alberi rosso-neri

- Definiamo **black-height** (bh) di un nodo x il numero di nodi neri lungo un qualunque percorso da x ad una foglia
  - Dal conteggio è escluso x ed è incluso il nodo foglia esterno

- La black-height di un albero rosso-nero è la black-height della radice

Altezza di un albero rosso-nero

Un albero rosso-nero con n nodi <u>interni</u> ha altezza al più 2lg(n+1)

Dimostriamo prima che un sottoalbero con radice in un nodo x contiene almeno $2^{bh(x)}-1$ nodi <u>interni</u>

- Per induzione sull'altezza di x
- Altezza pari a 0: è un nodo foglia esterno e la tesi è vera perché $2^{bh(x)}-1 = 2^0-1 = 0$ e il sottoalbero con radice in un nodo foglia esterno non contiene nodi interni
- Passo induttivo
  - Un nodo x con altezza >0 è un nodo interno con due figli
  - Ciascun figlio ha una black-height bh(x) se è rosso, bh(x)-1 se è nero
  - Dato che l'altezza dei figli è minore di quella di x, si può applicare l'ipotesi induttiva, ovvero ogni figlio ha almeno $2^{bh(x)-1}-1$ nodi interni
  - x avrà almeno $(2^{bh(x)-1}-1)+(2^{bh(x)-1}-1)+1 = 2^{bh(x)}-1$ nodi interni

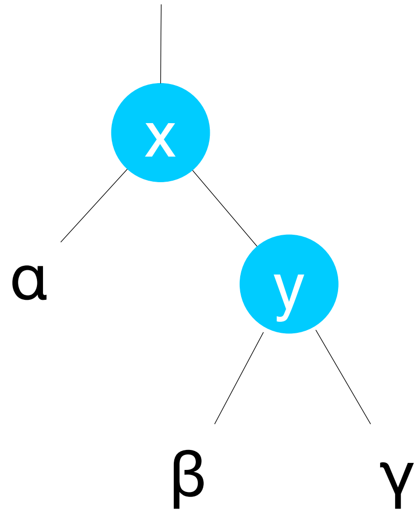
Altezza di un albero rosso-nero

Un albero rosso-nero con n nodi <u>interni</u> ha altezza al più 2lg(n+1)

- Per la proprietà 4, almeno la metà dei nodi su ogni percorso dalla radice ad una foglia, escludendo la radice, sono neri
- => La black-height bh della radice deve essere almeno h/2, dove h è l'altezza dell'albero ($h \leq 2*bh$)
- Il numero di nodi interni n è almeno $2^{h/2}-1$
- $n \geq 2^{bh}-1 \geq 2^{h/2}-1 \Leftrightarrow \lg(n+1) \geq h/2 \Leftrightarrow h \leq 2\lg(n+1)$
- Le query Minimum, Maximum, Successor e Predecessor richiedono un tempo O(lg n) su un albero rosso-nero
- Insert e Delete vanno riviste per preservare le proprietà dell'albero

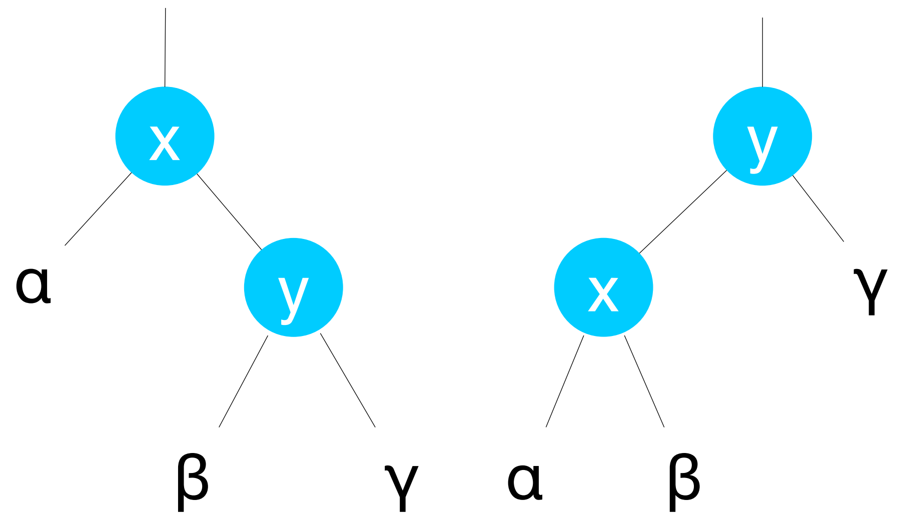
Rotazioni in alberi rosso-neri

- Per preservare le proprietà di un albero rosso-nero, vengono effettuate delle operazioni di rotazione
  - Preservano le proprietà di un albero di ricerca
- Rotazione a sinistra (sul nodo x)
  - Presuppone che il figlio di destra non sia NIL



1. β diventa figlio di destra di x
2. Il padre di x diventa il padre di y
3. x diventa il figlio di sinistra di y

Rotazioni in alberi rosso-neri

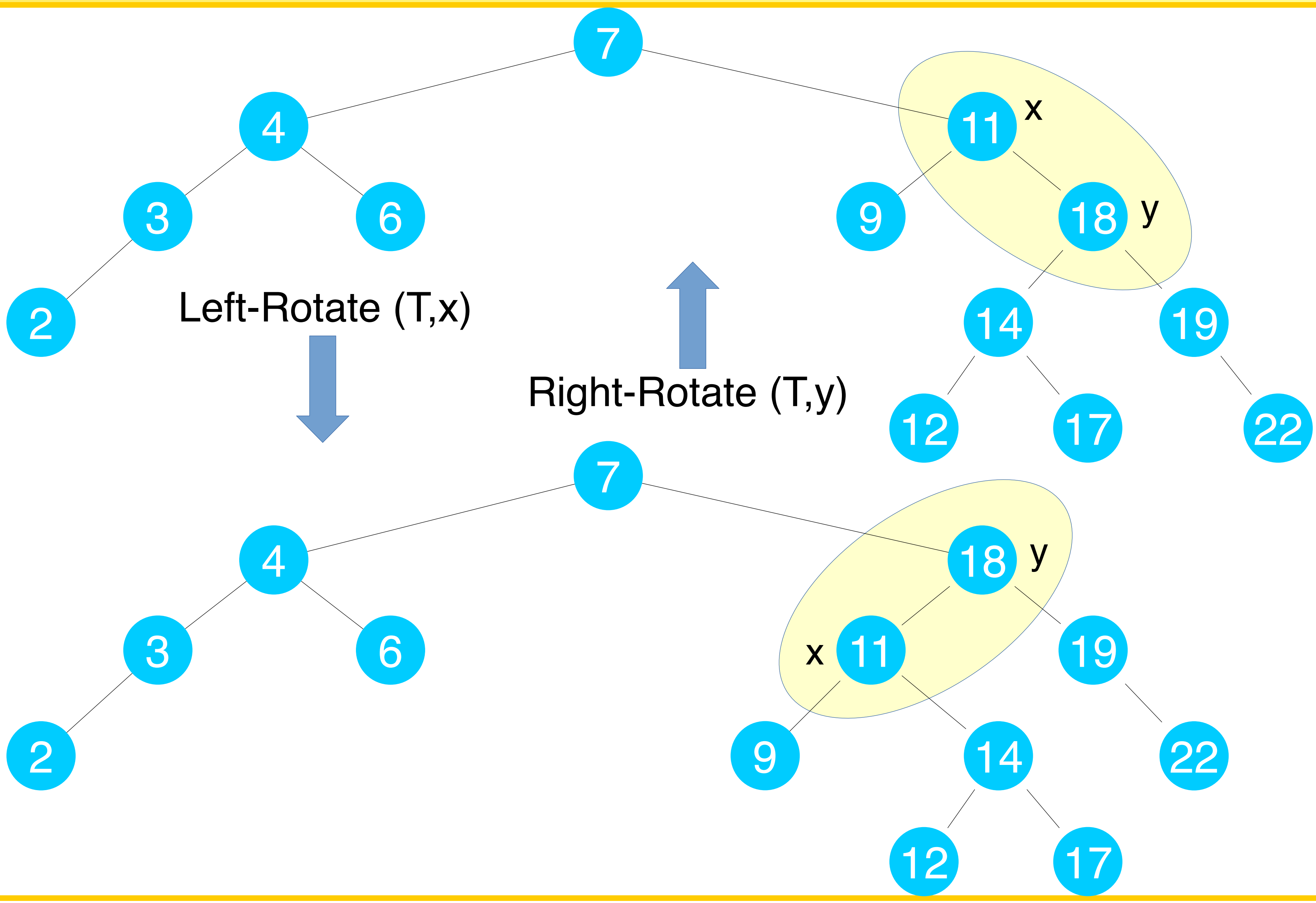


1. β diventa figlio di destra di x
2. Il padre di x diventa il padre di y
3. x diventa il figlio di sinistra di y

Il tempo di esecuzione è O(1)

```
Left-Rotate (T,x)
y ← right[x]
// operazione 1.
right[x] ← left[y]
if left[y]≠nil[T]
        then p[left[y]] ← x
// operazione 2.
p[y] ← p[x]
if p[x] = nil[T]
        then root[T] ← y
        else if x = left[p[x]]
            then left[p[x]] ←  y
            else right[p[x]] ← y
// operazione 3.
left[y] ← x
p[x] ← y
```

Rotazioni in alberi rosso-neri

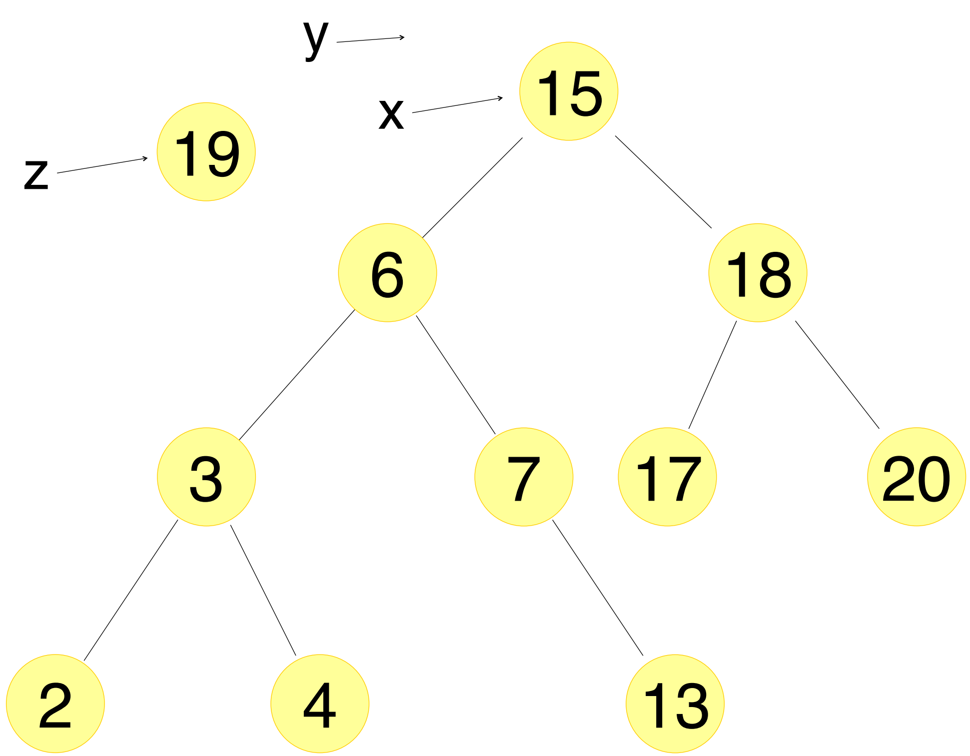


Inserimento di un nodo

- Strategia
  - Si inserisce il nodo come in un normale albero di ricerca, utilizzando una versione (leggermente) modificata di **Tree-Insert**
  - Si colora il nodo di rosso
  - Si invoca una funzione ausiliaria (**RB-Insert-Fixup**) per ripristinare le proprietà degli alberi rosso-neri

- Il tempo di esecuzione vedremo che è O(lg n)

Inserimento di un elemento




```
RB-Insert (T,z)
y ← nil[T]
x ← root[T]
while x≠nil[T]
        do y ← x
        if key[z] < key[x]
            then x ← left[x]
            else x ← right[x]
p[z] ← y
if y = nil[T]
        then root[T] ← z  // albero vuoto
        else if key[z] < key[y]
            then left[y] ← z
            else right[y] ← z
left[z] ← nil[T]
right[z] ←  nil[T]
color[z] ← RED
RB-Insert-Fixup (T,z)
```

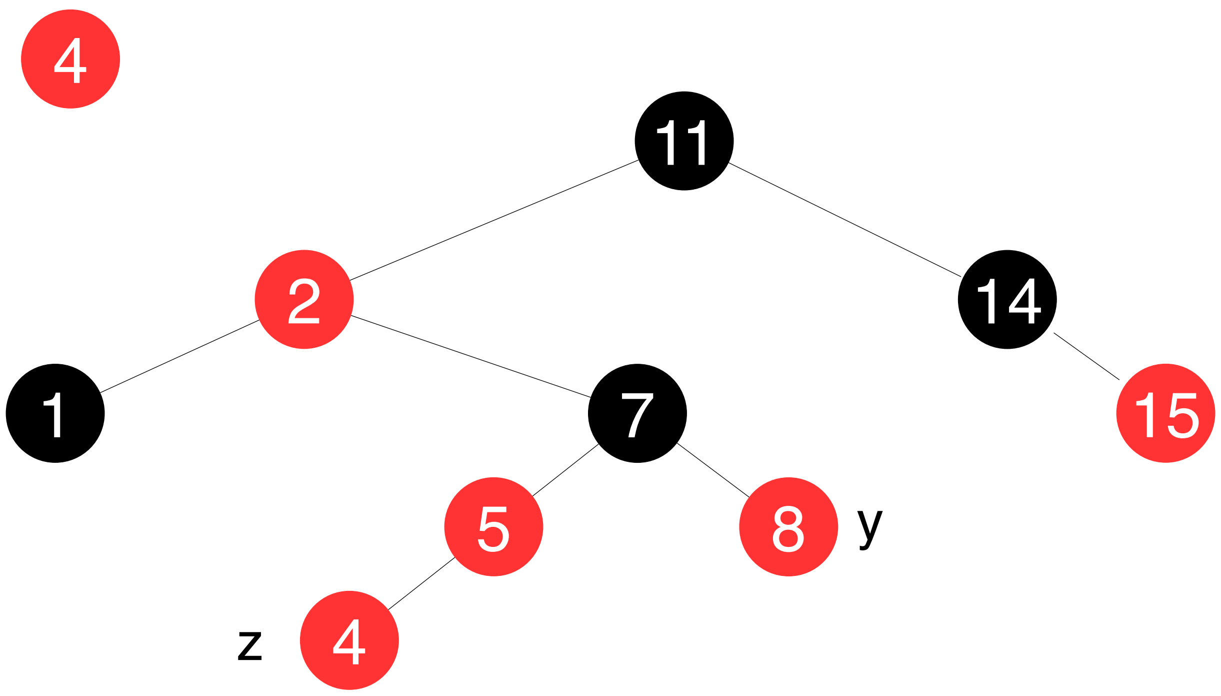
Il tempo di esecuzione è O(h)

Inserimento di un elemento

Quali delle proprietà degli alberi rosso-neri possono essere invalidate dall'inserimento di un nodo come visto in precedenza?

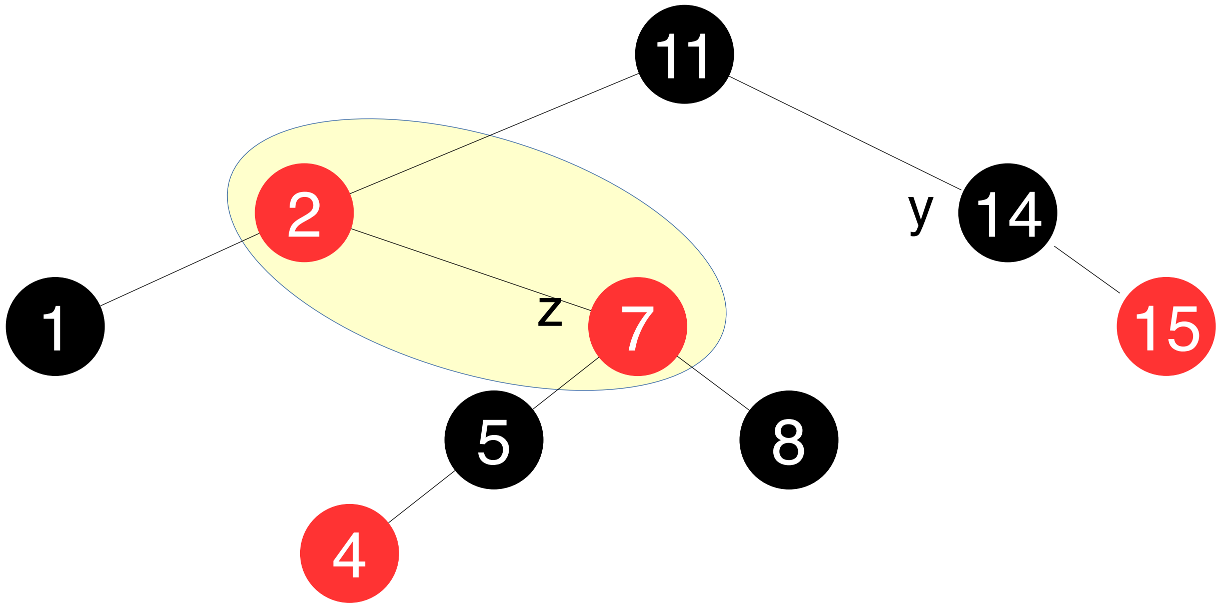
1. Ogni nodo o è rosso o è nero — VALE
2. La radice è nera — NON vale se l'albero era vuoto
3. Ogni nodo foglia esterno (NIL) è nero — VALE
4. Se un nodo è rosso, entrambi i figli sono neri — NON vale se il padre di z è rosso
5. Per ogni nodo, tutti i percorsi dal nodo alle foglie contengono lo stesso numero di nodi neri — VALE

Inserimento di un elemento

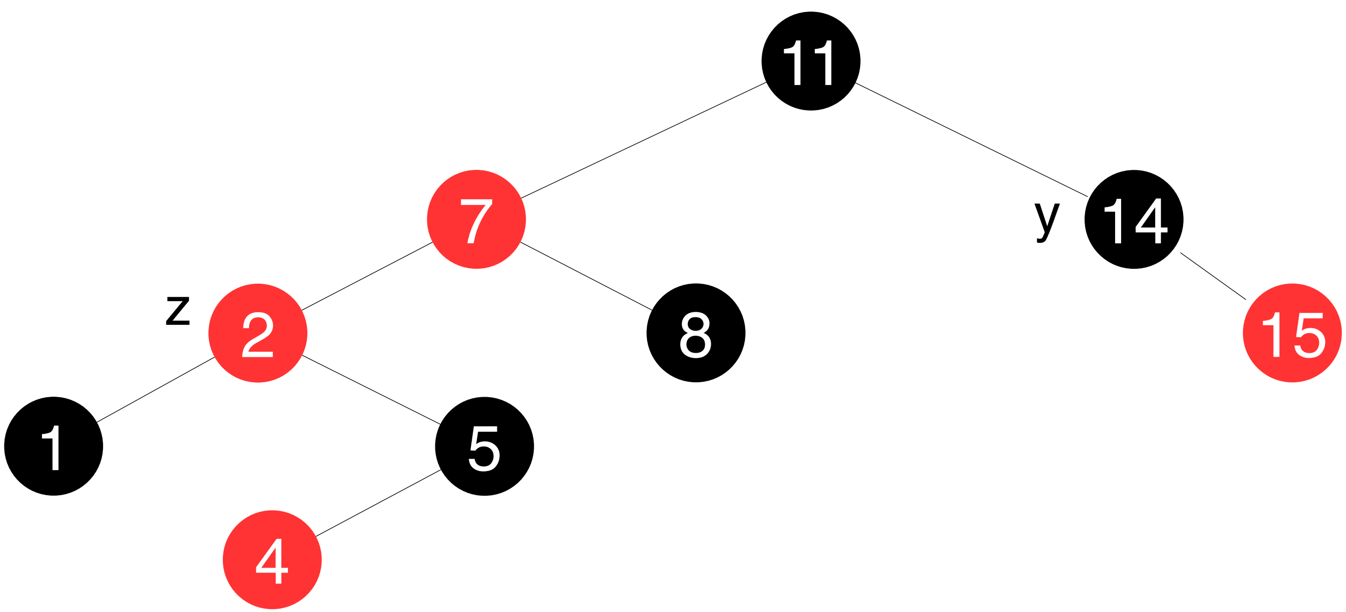


- CASO 1)
  - Violazione proprietà 4
  - Lo zio di z (y) è rosso
- I nodi sono ricolorati
- z diventa il nonno di z

Inserimento di un elemento

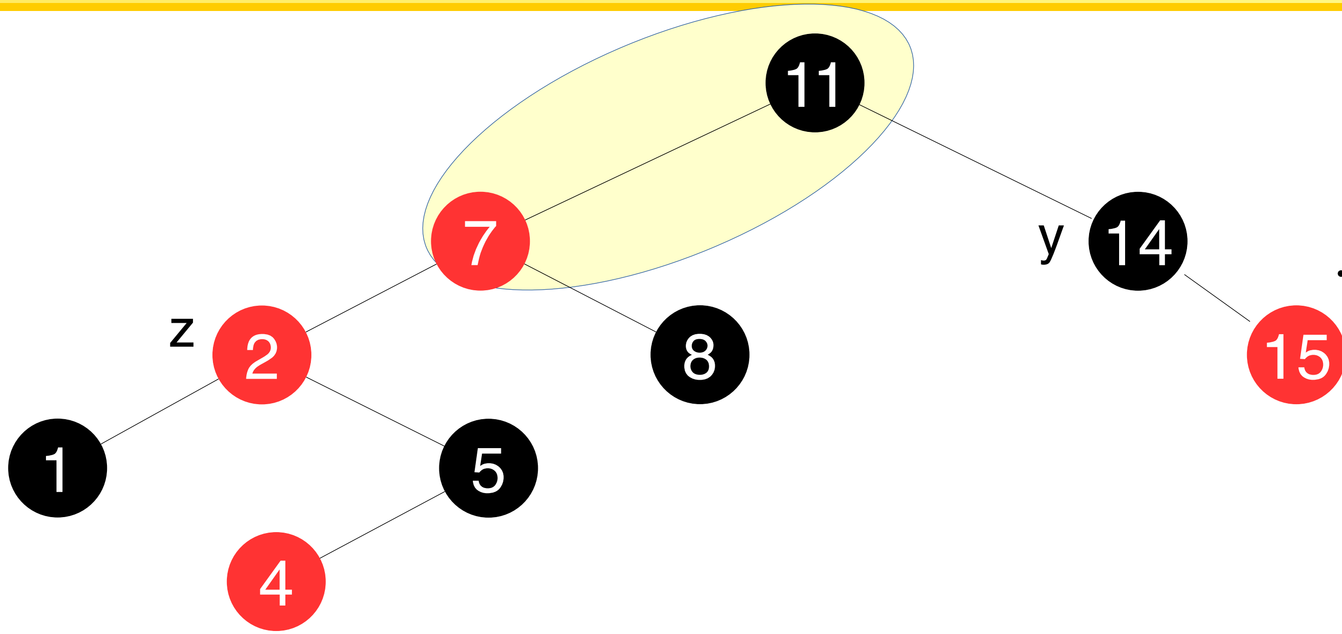


- CASO 2)
  - Violazione proprietà 4
  - Lo zio di z (y) è nero
  - z è il figlio di destra di p[z]

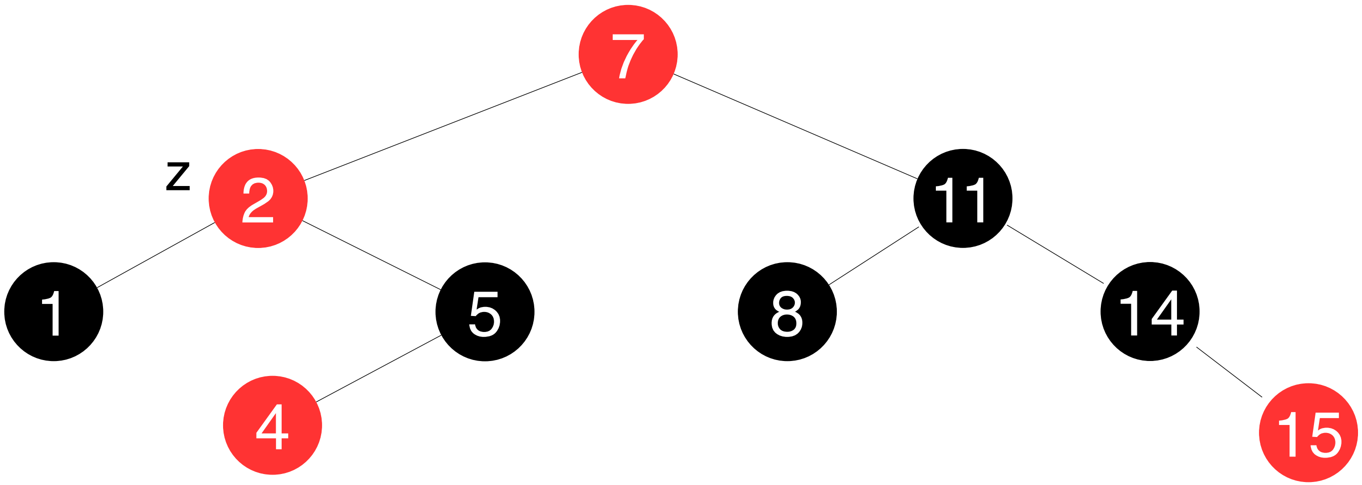


Rotazione a sinistra

Inserimento di un elemento

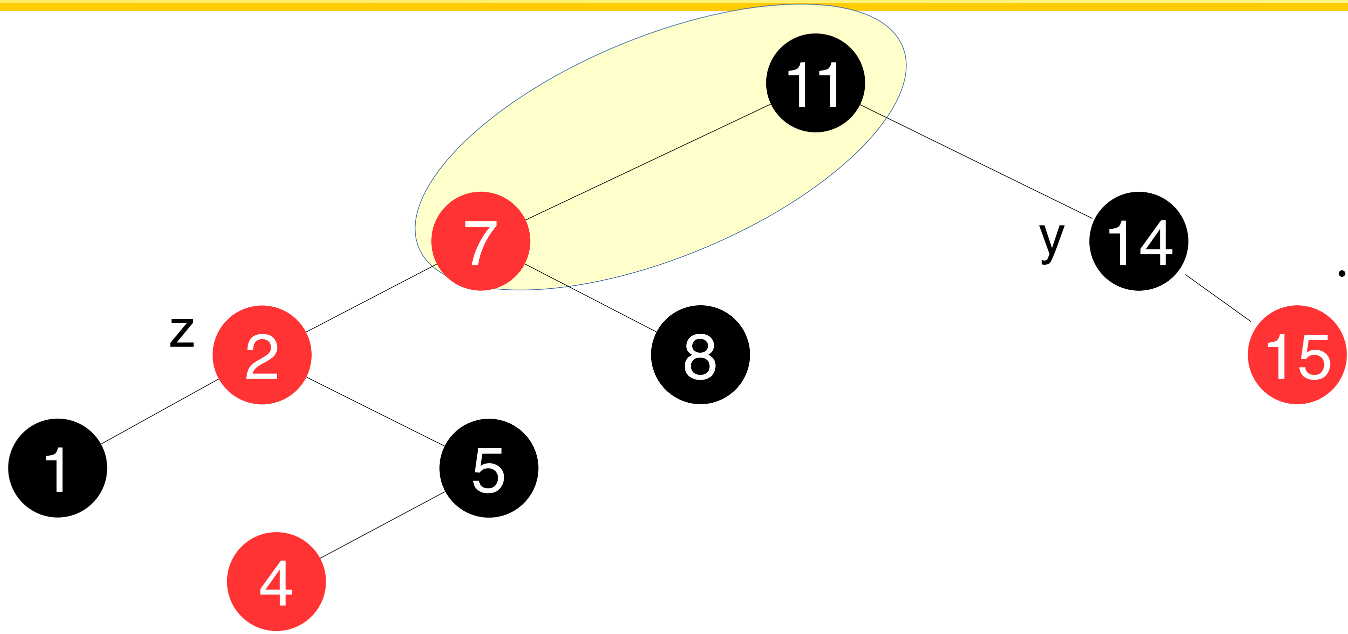


- CASO 3)
  - Violazione proprietà 4
  - Lo zio di z (y) è nero
  - z è il figlio di sinistra di p[z]

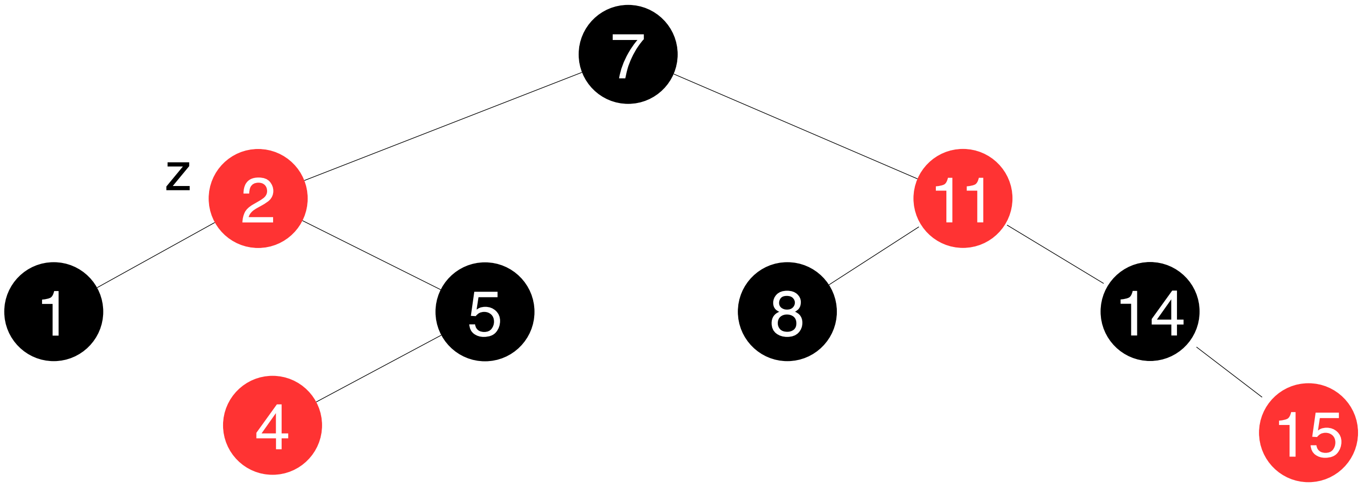


Rotazione a destra

Inserimento di un elemento



- CASO 3)
  - Violazione proprietà 4
  - Lo zio di z (y) è nero
  - z è il figlio di sinistra di p[z]



Rotazione a destra

Ricolora i nodi 7 e 11

Inserimento di un elemento

- RB-Insert-Fixup è costituito da un ciclo while, che viene eseguito fintantoché il padre di z resta rosso
- Costruito per mantenere il seguente invariante
- All'inizio di ogni iterazione del ciclo:
  - Il nodo z è rosso
  - Se p[z] è la radice, allora p[z] è nero
  - In tutto l'albero c'è al massimo una violazione delle proprietà degli alberi rosso-neri, ed è o di tipo 2 o di tipo 4
    - Se è di tipo 2, si presenta perché z è la radice ed è rosso
    - Se è di tipo 4, si presenta perché sia z che p[z] sono rossi

Inserimento di un elemento

a) Il nodo z è rosso
b) Se p[z] è la radice, allora p[z] è nero
c) In tutto l'albero c'è al massimo una violazione delle proprietà degli alberi rosso-neri, ed è o di tipo 2 o di tipo 4
   1. Se è di tipo 2, si presenta perché z è la radice ed è rosso
   2. Se è di tipo 4, si presenta perché sia z che p[z] sono rossi

- Prima della prima iterazione (z è il nodo che è stato inserito):
  a) Vera perché il nodo inserito è di colore rosso
  b) Se p[z] è la radice, non è stata modificata da RB-Insert e quindi è rimasta nera
  c) Le proprietà 1, 3 e 5 non vengono violate da RB-Insert
     1. Se l'albero era vuoto e il nodo inserito è la nuova radice, c'è una violazione di tipo 2. Non ci sono violazioni di tipo 4 (foglie nere)
     2. Se l'albero non era vuoto e z viene inserito come figlio di un nodo rosso, c'è una violazione di tipo 4. Non ci sono violazioni di tipo 2

Inserimento di un elemento

a) Il nodo z è rosso
b) Se p[z] è la radice, allora p[z] è nero
c) In tutto l'albero c'è al massimo una violazione delle proprietà degli alberi rosso-neri, ed è o di tipo 2 o di tipo 4

1. Se è di tipo 2, si presenta perché z è la radice ed è rosso
2. Se è di tipo 4, si presenta perché sia z che p[z] sono rossi

- Il ciclo termina quando il padre di z è nero:

=> L'invariante (c.2) garantisce che non ci sono violazioni di tipo 4

- In base a c.1), ci potrebbe essere una violazione di tipo 2 dove z è la radice ed è rosso
- È sufficiente ricolorare di nero la radice

- Nota: se z è il primo nodo inserito, il ciclo while termina subito
  - Il padre di z è la sentinella, che è un nodo nero

Inserimento di un elemento

- Vediamo come implementare il corpo del ciclo while in modo da preservare l'invariante
- Se siamo entrati nel ciclo while per effettuare un'iterazione:
    - Il padre di z (p[z]) è rosso
    - Il nonno di z (p[p[z]]) esiste (è un nodo interno)
        - Dato che p[z] è rosso, per b) non può essere la radice e quindi il padre di p[z] non è la sentinella ma un nodo interno
    - Il nonno di z (p[p[z]]) è un nodo nero
        - Se p[p[z]] fosse rosso, dato che p[z] è rosso, avremmo una violazione di tipo 4 tra p[z] e p[p[z]], contrariamente a c.2)
    - Non ci sono violazioni di tipo 2
        - Se ci fosse una violazione di tipo 2, z sarebbe la radice e il padre di z (la sentinella) sarebbe nero (assurdo: p[z] è rosso)

Inserimento di un elemento

- Vediamo come implementare il corpo del ciclo while in modo da preservare l'invariante
- C'è da distinguere i casi in cui il padre di z è figlio di sinistra o figlio di destra (del nonno di z)
  - I due casi sono simmetrici (si scambia left con right e viceversa)
  - Consideriamo il caso in cui il padre di z è il figlio di sinistra

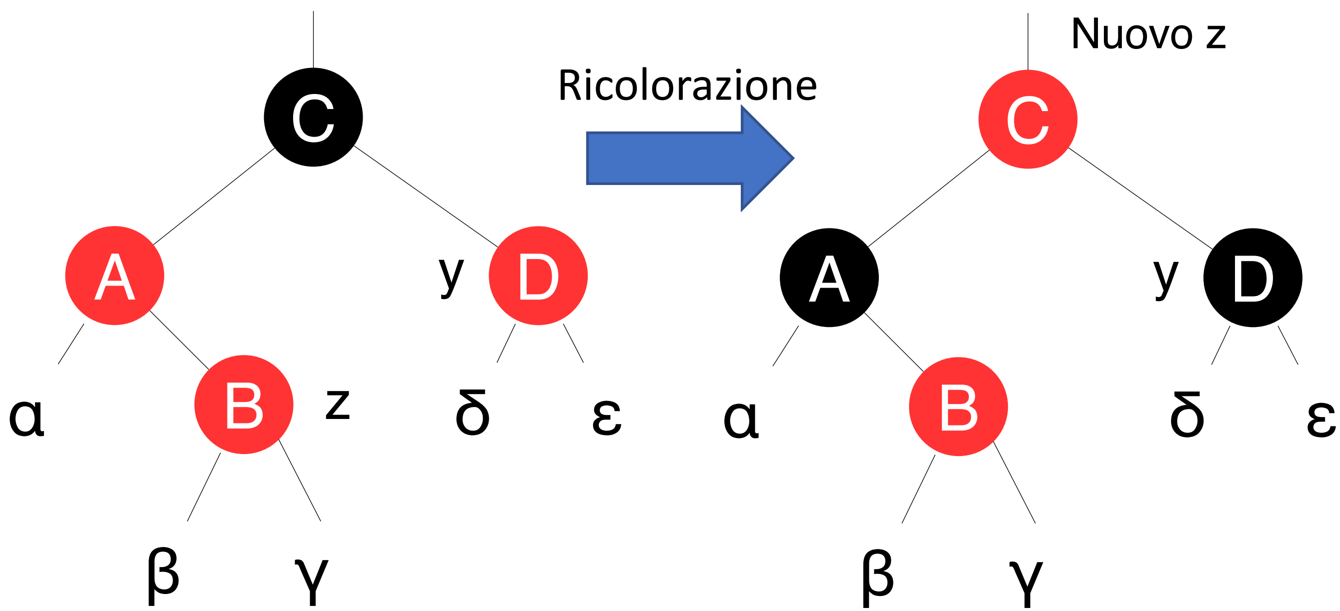
```
RB-Insert-Fixup (T,z)
while color[p[z]] = RED
        do if p[z] = left[p[p[z]]]
              then y ← right[p[p[z]] // zio




color[root[T]] ← BLACK
```

Inserimento di un elemento

- Sappiamo che z è rosso, p[z] è rosso e p[p[z]] è nero
- Distinguiamo sulla base del colore dello zio (y)
- Caso 1) y è rosso

<u>RB-Insert-Fixup</u> (T,z)

```
while color[p[z]] = RED
        do if p[z] = left[p[p[z]]]
            then y ← right[p[p[z]] // zio
                if color[y] = RED
                    then color[p[z]]←BLAC
                         color[y] ← BLACK
                         color[p[p[z]]←RE
                         z ← p[p[z]]
```




```
color[root[T]] ← BLACK
```

Inserimento di un elemento

a) Il nodo z è rosso
b) Se p[z] è la radice, allora p[z] è nero
c) In tutto l'albero c'è al massimo una violazione delle proprietà degli alberi rosso-neri, ed è o di tipo 2 o di tipo 4

1. Se è di tipo 2, si presenta perché z è la radice ed è rosso
2. Se è di tipo 4, si presenta perché sia z che p[z] sono rossi

- Verifichiamo l'invariante
  - z' è il valore di z nella nuova iterazione

a) z' è p[p[z]], che è colorato di rosso

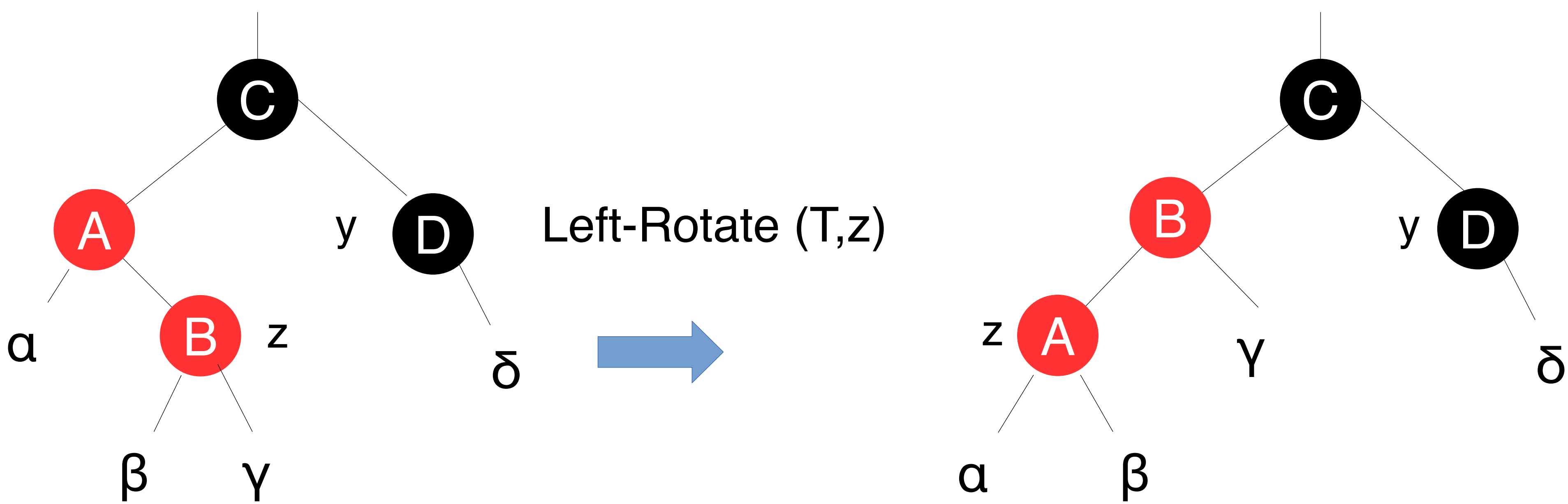
b) p[z'] non è modificato in questa iterazione, quindi se è la radice allora è nero (no violazioni di tipo 2 se entriamo nel while)

c) Non introduciamo violazioni di tipo 5 (né 1 e 3)

1. Se z' è la radice abbiamo risolto la violazione di tipo 4 e resta una sola violazione di tipo 2 (z' è la radice ed è rosso)
2. Se z' non è la radice, non abbiamo introdotto violazione di tipo 2. Ci può essere una violazione di tipo 4 tra z' e p[z'] se p[z'] è rosso

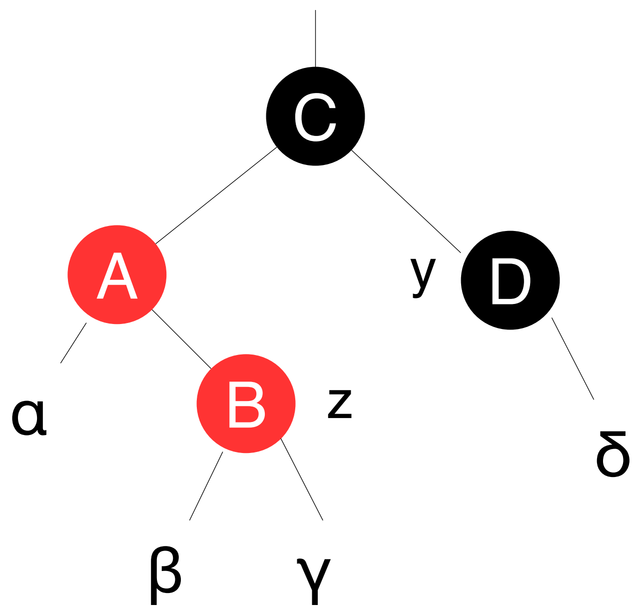
Inserimento di un elemento

- Caso 2) y è nero e z è figlio di destra
- z ← p[z] e Left-Rotate su z
  - Dato che z e p[z] sono rossi, non si viola proprietà 5 e black-height
  - Ci riconduciamo al caso 3



Inserimento di un elemento

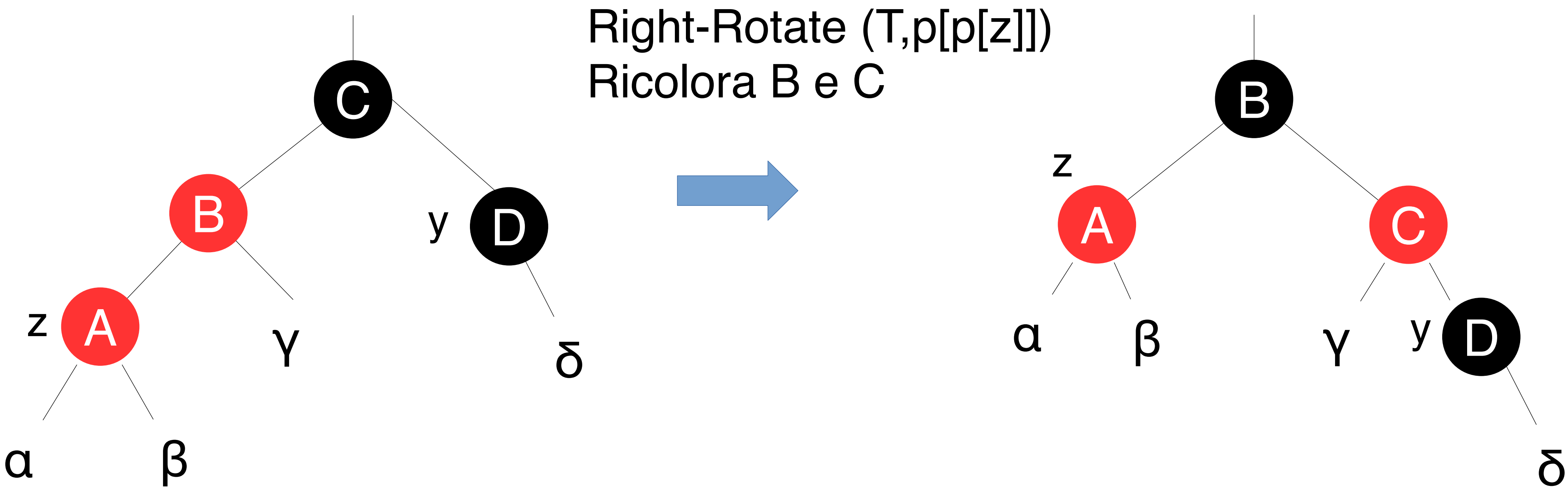
- Caso 2) y è nero e z è figlio di destra
  - z ← p[z] e Left-Rotate su z




```
RB-Insert-Fixup (T,z)
while color[p[z]] = RED
        do if p[z] = left[p[p[z]]]
                then y ← right[p[p[z]] // zio
                        if color[y] = RED
                                then color[p[z]]←BLAC
                                        color[y] ← BLACK
                                        color[p[p[z]]←RED
                                        z ← p[p[z]]
                        else if z = right[p[z]]
                                then z ← p[z]
                                        Left-Rotate(T,z)

color[root[T]] ← BLACK
```

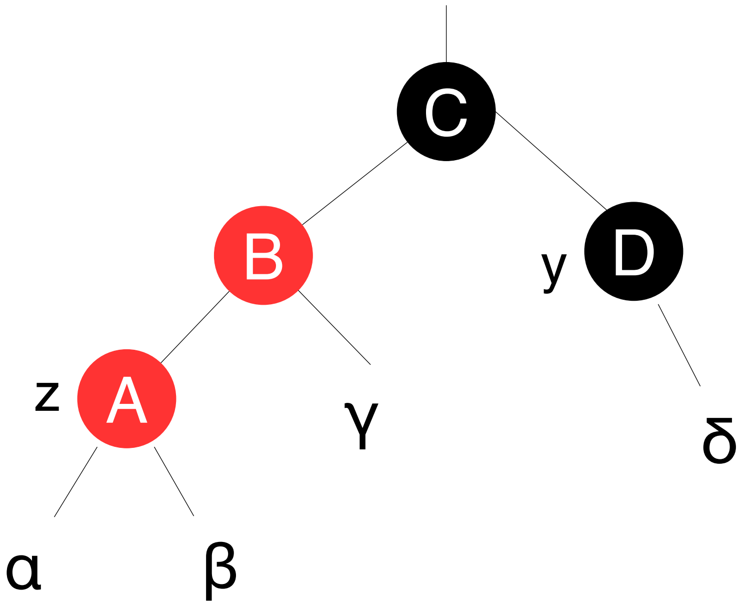
Inserimento di un elemento

- Caso 3) y è nero e z è figlio di sinistra
- p[z] diventa nero, p[p[z]] rosso e Right-Rotate su p[p[z]]
  - preservano la proprietà 5
  - Il ciclo termina perché p[z] è nero



Inserimento di un elemento

- Caso 3) y è nero e z è figlio di sinistra
- p[z] diventa nero, p[p[z]] rosso e Right-Rotate su p[p[z]]




```
RB-Insert-Fixup (T,z)
while color[p[z]] = RED
        do if p[z] = left[p[p[z]]]
              then y ← right[p[p[z]] // zio
                 if color[y] = RED
                    then color[p[z]]←BLACK
                         color[y] ← BLACK
                         color[p[p[z]]←RED
                         z ← p[p[z]]
                 else if z = right[p[z]]
                    then z ← p[z]
                         Left-Rotate(T,z)
                    color[p[z]]←BLACK
                    color[p[p[z]] ← RED
                    Right-Rotate(T,p[p[z]])
           else (stesso codice di then
                con right e left scambiati)
color[root[T]] ← BLACK
```

Inserimento di un elemento

a) Il nodo z è rosso
b) Se p[z] è la radice, allora p[z] è nero
c) In tutto l'albero c'è al massimo una violazione delle proprietà degli alberi rosso-neri, ed è o di tipo 2 o di tipo 4

1. Se è di tipo 2, si presenta perché z è la radice ed è rosso
2. Se è di tipo 4, si presenta perché sia z che p[z] sono rossi

a) Nel caso 2, z diventa p[z]], che è rosso, e poi non si modifica più z
b) Il caso 3 colora p[z] di nero
c) Non introduciamo violazioni di tipo 5 (né 1 e 3)
    1. z (che è rosso) non è la radice e quindi non viola 2. Non ci sono altre violazioni di 2 perché l'unico nodo che diventa rosso diventa figlio di un nodo nero (e quindi non può essere la radice)
    2. I casi 2 e 3 correggono una violazione di 4 senza introdurne altre

Inserimento di un elemento

- Tempo di esecuzione di RB-Insert-Fixup è O(lg n)
  - Il ciclo si ripete solo se siamo nel caso 1, durante il quale z sale di due livelli
  - Il ciclo viene ripetuto O(lg n) volte

- Vengono eseguite al più due rotazioni
  - Il ciclo termina quando entriamo nel caso 2 o 3

Eliminazione di un elemento

- Anche l'eliminazione di un elemento richiede O(lg n)
- Si invoca la Tree-Delete modificata:
  - I riferimenti a NIL si sostituiscono con nil[T]
  - Nella Tree-Delete, l'assegnazione al padre di x del padre di y (fatta per tagliare y), si fa solo se x non è nullo (y è una foglia). Nella RB-Delete, questa assegnazione si fa sempre
    - Se x è la sentinella nil[T], il padre della sentinella diventa il padre del nodo tagliato fuori
  - Se il nodo tagliato fuori è nero, si chiama RB-Delete-Fixup

- Esegue rotazioni e cambiamenti di colore per ripristinare le proprietà di un albero rosso-nero

Eliminazione di un elemento

- Un nodo rosso o è una foglia o ha due figli (neri)
  - Altrimenti è violata la proprietà 5
- L'unico caso in cui il nodo tagliato fuori è rosso è il caso 1 (z è una foglia rossa)
- In questo caso, l'albero è RB
  - Nessuna black-height cambia (proprietà 5)
  - Non si creano adiacenze tra nodi rossi (proprietà 4)
  - y, essendo rosso, non poteva essere la radice, che resta nera (proprietà 2)