ALBERI

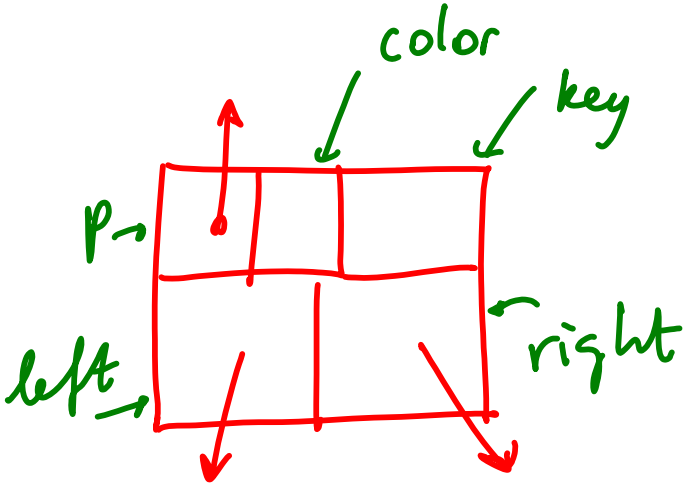
ROSSO-NERI

- GLI ALBERI ROSSO-NERI RAPPRESENTANO UNA VARIANTE "BILANCIATA" DEGLI ALBERI BINARI DI RICERCA

- I NODI DI UN ALBERO ROSSO-NERO CONTENGONO I SEGUENTI ATTRIBUTI:

  LEFT (PUNTATORE AL FIGLIO SINISTRO / NIL)
  RIGHT (PUNTATORE AL FIGLIO DESTRO / NIL)
  P (PUNTATORE AL PADRE / NIL)
  KEY (CHIAVE)
  COLOR (ROSSO/NERO)

  + EVENTUALI CAMPI SATELLITI



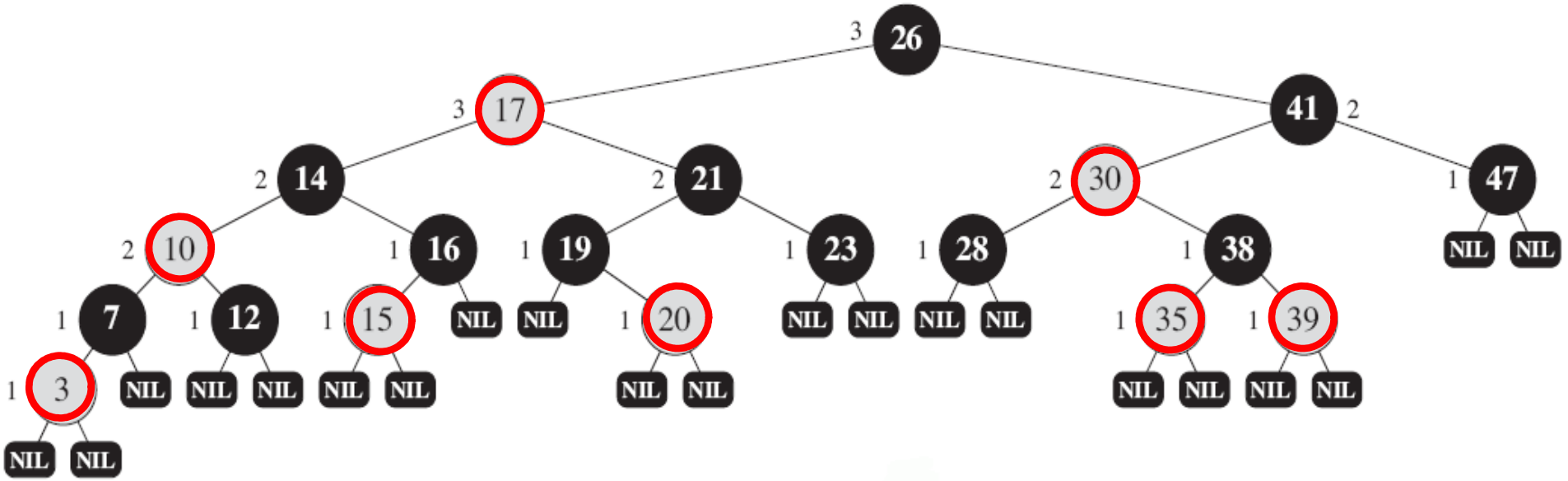
- SE MANCA UN FIGLIO O IL PADRE DI UN NODO, IL CORRISPONDENTE PUNTATORE CONTERRÀ IL VALORE NIL

- TRATTEREMO TALI VALORI NIL COME FOGLIE ESTERNE

- I RIMANENTI NODI (QUELLI CONTENENTI CHIAVI) SARANNO TRATTATI COME NODI INTERNI

- UN ALBERO ROSSO-NERO E' UN ALBERO BINARIO DI RICERCA CHE SODDISFA LE SEGUENTI PROPRIETA':
  - OGNI NODO E' ROSSO O NERO
  - LA RADICE E' NERA
  - OGNI FOGLIA (NIL) E' NERA
  - SE UN NODO E' ROSSO, ALLORA ENTRAMBI I SUOI FIGLI SONO NERI
  - PER OGNI NODO, TUTTI I CAMMINI SEMPLICI CHE VANNO DAL NODO A CIASCUNA SUA FOGLIA (NIL) DISCENDENTE CONTENGONO LO STESSO NUMERO DI NODI NERI (PROPRIETA' DI BILANCIAMENTO)

ESEMPIO:



DEF. ALTEZZA NERA DI UN NODO x (bh(x))

E' IL NUMERO DI NODI NERI IN UN QUALUNQUE CAMMINO DA x (CON L'ESCLUSIONE DI x) SINO AD UNA SUA FOGLIA (NIL) DISCENDENTE

LEMMA IL SOTTOALBERO RADICATO IN UN NODO x DI UN ALBERO ROSSO-NERO CONTIENE ALMENO $2^{bh(x)} - 1$ NODI INTERNI.

DIM. PER INDUZIONE SULL'ALTEZZA h DI x.

CASO BASE: $h = 0$

- x E' UNA FOGLIA (NIL)
- QUINDI IL SOTTOALBERO RADICATO IN x CONTIENE 0 NODI INTERNI
- $bh(x) = 0$, $2^{bh(x)} - 1 = 2^0 - 1 = 1 - 1 = 0$

PASSO INDUTTIVO: $h > 0$

- x È UN NODO INTERNO CON DUE FIGLI
- CIASCUN FIGLIO DI x HA UN'ALTEZZA NERA PARI A
  - $bh(x)$, SE IL SUO COLORE È ROSSO
  - $bh(x)-1$, SE IL SUO COLORE È NERO
- PER L'IPOTESI INDUTTIVA, CIASCUNO DEI DUE ALBERI RADICATI NEI FIGLI DI x CONTIENE ALMENO $2^{bh(x)-1}-1$ NODI INTERNI
- QUINDI IL SOTTOALBERO RADICATO IN x CONTIENE ALMENO

$$\left(2^{bh(x)-1}-1\right) + \left(2^{bh(x)-1}-1\right) + 1 = 2^{bh(x)}-1$$

NODI INTERNI.

■

LEMMA L'ALTEZZA DI UN ALBERO ROSSO-NERO CON n NODI INTERNI E' AL PIÙ $2 \lg(n+1)$.

DIM. SIA h L'ALTEZZA DELL'ALBERO.

- ALMENO LA METÀ DEI NODI IN UN QUALSIASI CAMMINO SEMPLICE DALLA RADICE AD UNA SUA FOGLIA (NIL), ESCLUSA LA RADICE, E' NERA
- QUINDI $bh(root) \geq h/2$.
- $n \geq 2^{bh(root)} - 1 \geq 2^{h/2} - 1$ (PER IL LEMMA PRECEDENTE)
- PERTANTO: $h \leq 2 \lg(n+1)$. ■

COROLLARIO IN UN ALBERO ROSSO-NERO CON n NODI INTERNI, LE OPERAZIONI:

- SEARCH
- MINIMUM
- MAXIMUM
- PREDECESSOR
- SUCCESSOR

POSSONO ESSERE IMPLEMENTATE NEL TEMPO $O(\lg n)$. ■

- OCCORRERÀ MODIFICARE OPPORTUNAMENTE LE OPERAZIONI TREE-INSERT E TREE-DELETE AFFINCHÉ LE PROPRIETÀ DEGLI ALBERI ROSSO-NERI SIANO MANTENUTE.

ESERCIZI

13.1-1

In the style of Figure 13.1(a), draw the complete binary search tree of height 3 on the keys $\{1, 2, \ldots, 15\}$. Add the NIL leaves and color the nodes in three different ways such that the black-heights of the resulting red-black trees are 2, 3, and 4.

13.1-2

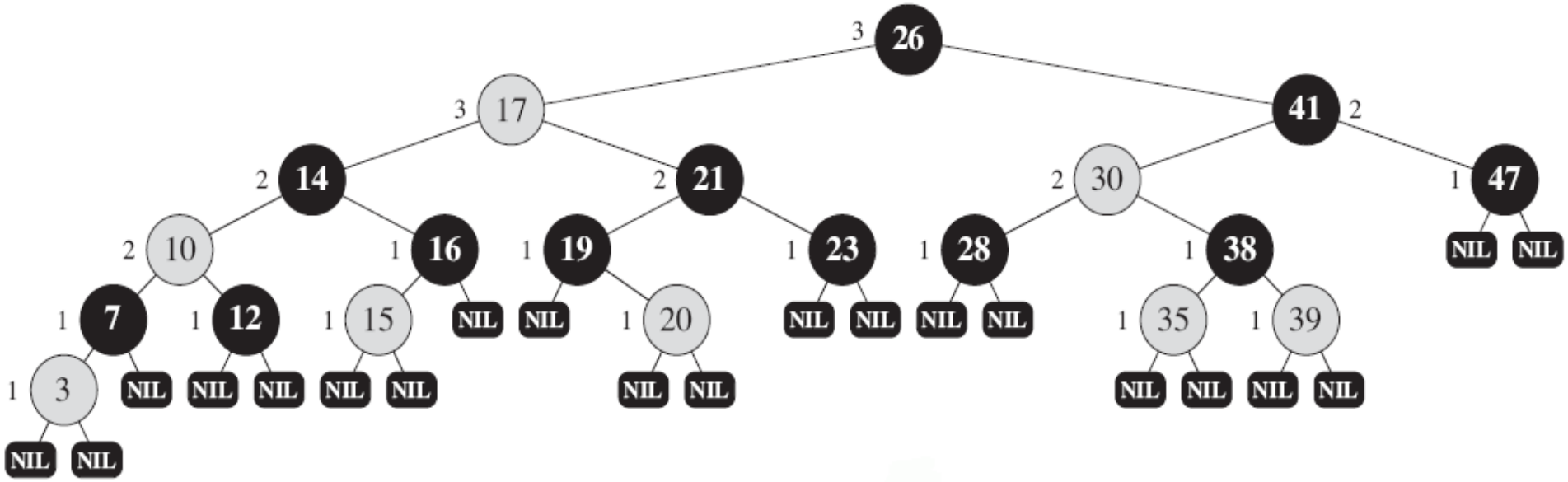
Draw the red-black tree that results after TREE-INSERT is called on the tree in Figure 13.1 with key 36. If the inserted node is colored red, is the resulting tree a red-black tree? What if it is colored black?

13.1-3

Let us define a ***relaxed red-black tree*** as a binary search tree that satisfies red-black properties 1, 3, 4, and 5. In other words, the root may be either red or black. Consider a relaxed red-black tree T whose root is red. If we color the root of T black but make no other changes to T, is the resulting tree a red-black tree?

13.1-4

Suppose that we "absorb" every red node in a red-black tree into its black parent, so that the children of the red node become children of the black parent. (Ignore what happens to the keys.) What are the possible degrees of a black node after all its red children are absorbed? What can you say about the depths of the leaves of the resulting tree?



13.1-5

Show that the longest simple path from a node x in a red-black tree to a descendant leaf has length at most twice that of the shortest simple path from node x to a descendant leaf.

13.1-6

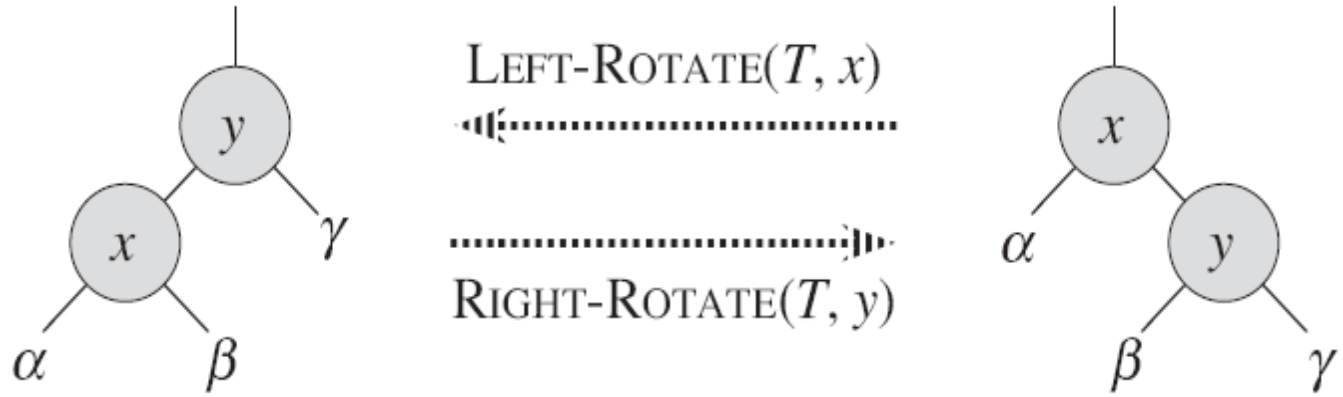
What is the largest possible number of internal nodes in a red-black tree with black-height k? What is the smallest possible number?

13.1-7

Describe a red-black tree on n keys that realizes the largest possible ratio of red internal nodes to black internal nodes. What is this ratio? What tree has the smallest possible ratio, and what is the ratio?

ROTAZIONE

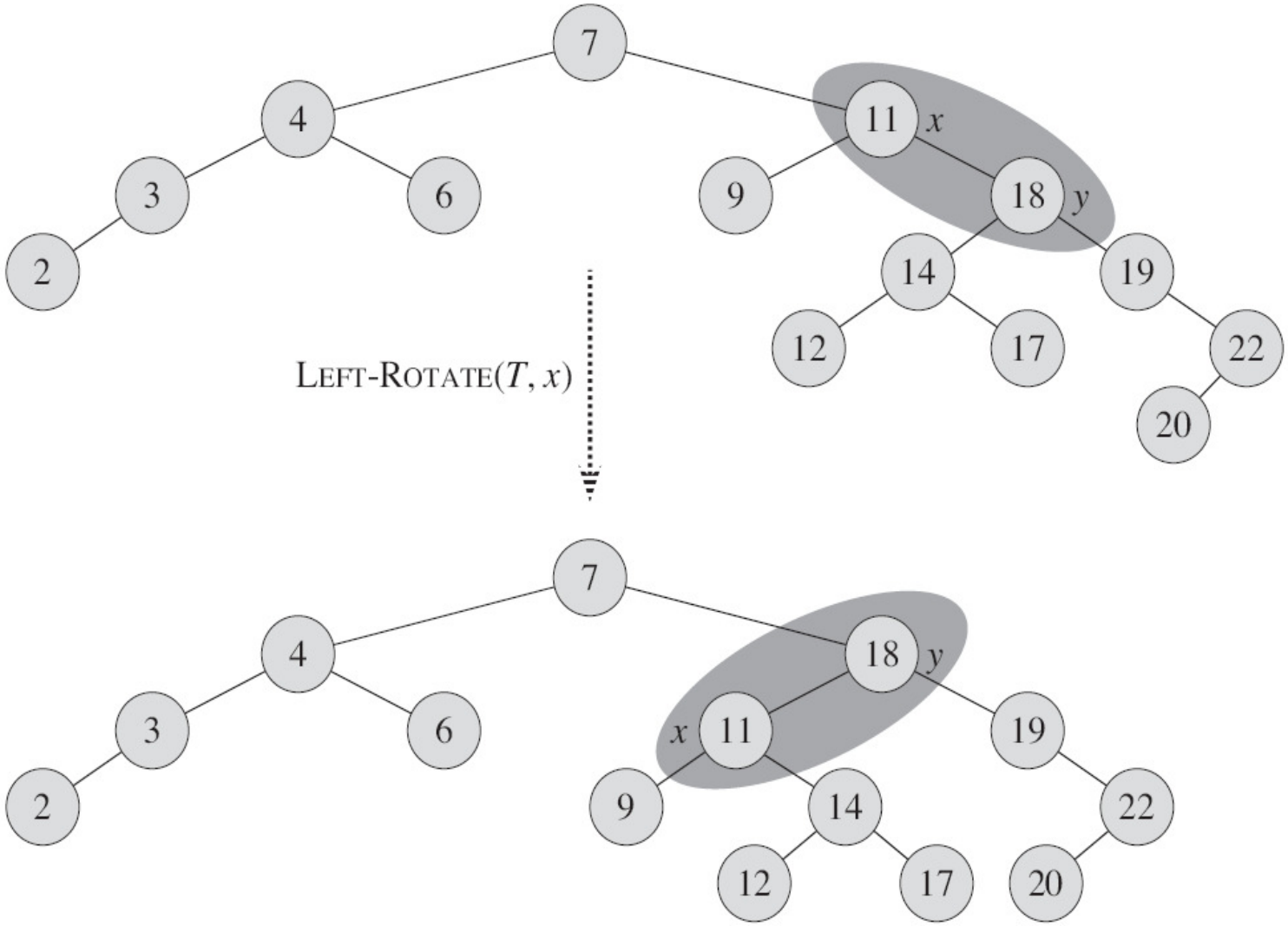
- PER RIPRISTINARE LE PROPRIETÀ DEGLI ALBERI ROSSO-NERI A SEGUITO DI INSERIMENTI E CANCELLAZIONI, SI UTILIZZERANNO OPPORTUNE ROTAZIONI CHE PRESERVANO LA PROPRIETÀ DEGLI ALBERI DI RICERCA.



LEFT-ROTATE(T, x)

```
 1  y = x.right              // set y
 2  x.right = y.left         // turn y's left subtree into x's right subtree
 3  if y.left ≠ T.nil
 4       y.left.p = x
 5  y.p = x.p                // link x's parent to y
 6  if x.p == T.nil
 7       T.root = y
 8  elseif x == x.p.left
 9       x.p.left = y
10  else x.p.right = y
11  y.left = x               // put x on y's left
12  x.p = y
```

ESEMPIO


7
4
11
x
3
6
9
18
y
2
14
19
12
17
22
20
LEFT-ROTATE(T, x)
7
4
18
y
3
6
x
11
19
2
9
14
22
12
17
20

ESERCIZI

13.2-2

Argue that in every n-node binary search tree, there are exactly $n - 1$ possible rotations.

13.2-3

Let a, b, and c be arbitrary nodes in subtrees α, β, and γ, respectively, in the left tree of Figure 13.2. How do the depths of a, b, and c change when a left rotation is performed on node x in the figure?

13.2-4

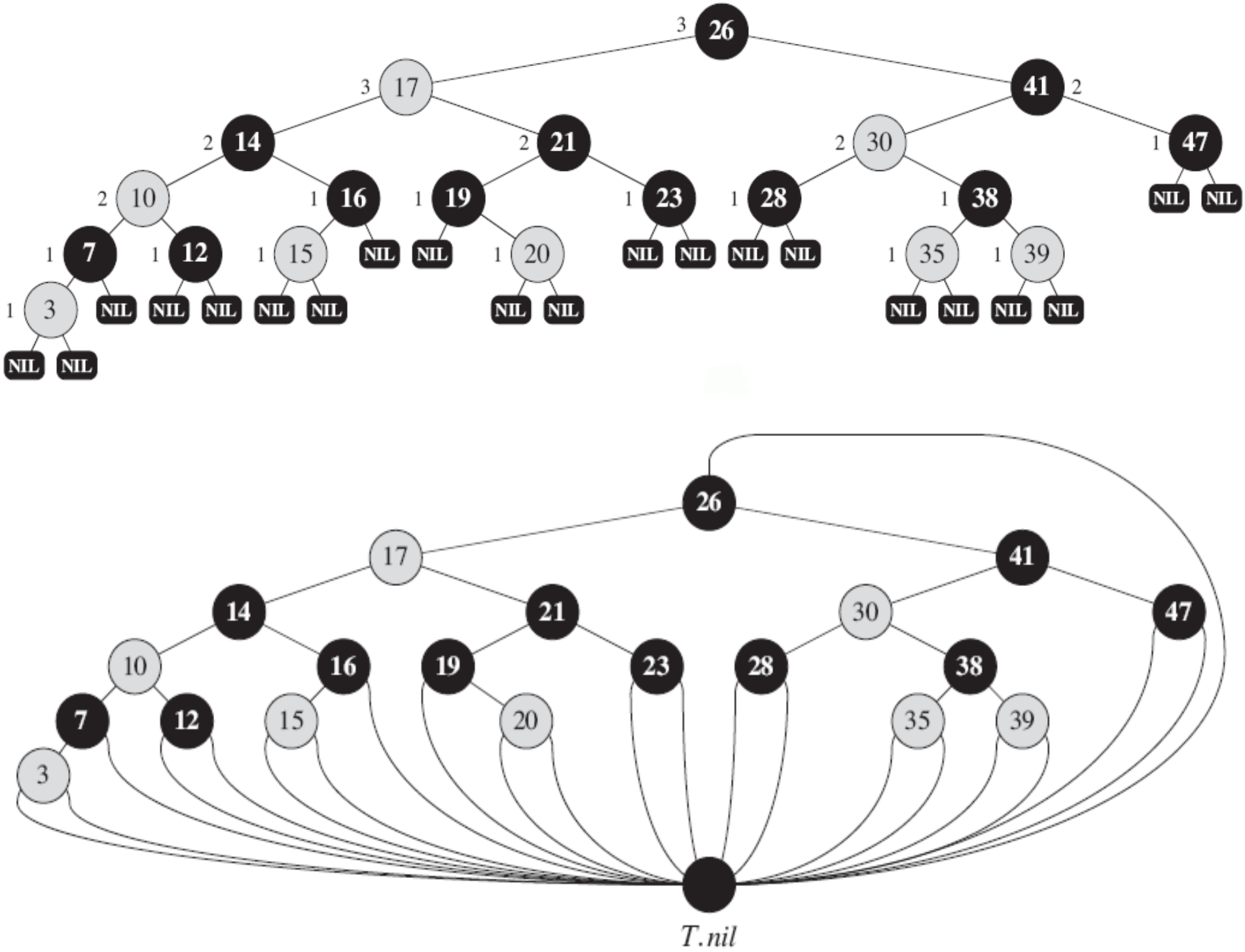
Show that any arbitrary n-node binary search tree can be transformed into any other arbitrary n-node binary search tree using $O(n)$ rotations. (*Hint:* First show that at most $n - 1$ right rotations suffice to transform the tree into a right-going chain.)

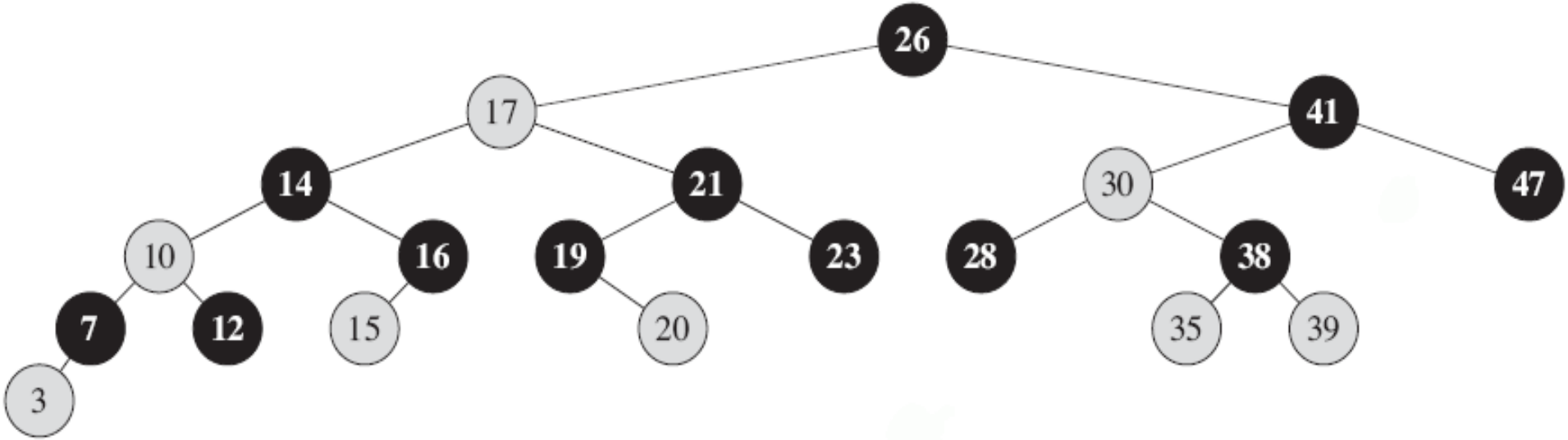
NIL vs. T.nil

- AL POSTO DEL VALORE NIL UTILIZZEREMO IL NODO T.nil

- SARÀ DUNQUE POSSIBILE USARE T.nil COME SENTINELLA

- I VALORI DEGLI ATTRIBUTI p, left, right, key DI T.nil SONO IRRILEVANTI, TUTTAVIA RISULTERÀ POSSIBILE ASSEGNARE AD ESSI DEI VALORI


3 26
3 17
41 2
2 14
2 21
2 30
1 47
2 10
1 16
1 19
1 23
1 28
1 38
NIL NIL
1 7
1 12
1 15
NIL
NIL
1 20
NIL NIL
NIL NIL
1 35
1 39
1 3
NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL
NIL NIL
26
17
41
14
21
30
47
10
16
19
23
28
38
7
12
15
20
35
39
3
T.nil


26
17
41
14
21
30
47
10
16
19
23
28
38
7
12
15
20
35
39
3

INSERIMENTO

```
RB-INSERT(T, z)
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == T.nil
10       T.root = z
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

```
TREE-INSERT(T, z)
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == NIL
10       T.root = z          // tree T was empty
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
```

DIFFERENZA 1 NIL ⇝ T.nil

RB-INSERT(T, z)

```
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == T.nil
10       T.root = z
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

TREE-INSERT(T, z)

```
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == NIL
10       T.root = z          // tree T was empty
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
```

DIFFERENZA 2 ASSEGNAMENTI ESPLICITI DI T.nil

RB-INSERT(T, z)

```
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == T.nil
10       T.root = z
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

TREE-INSERT(T, z)

```
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == NIL
10       T.root = z          // tree T was empty
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
```

DIFFERENZA 3 COLORAZIONE DI ROSSO DEL NODO z APPENA INSERITO

RB-INSERT(T, z)

```
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == T.nil
10       T.root = z
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

TREE-INSERT(T, z)

```
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == NIL
10       T.root = z          // tree T was empty
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
```

DIFFERENZA 4

CHIAMATA DI RB-INSERT-FIXUP(T, z) PER RIPRISTINARE, SE VENUTA MENO, LA PROPRIETÀ CHE VIETA DUE NODI ROSSI PADRE-FIGLIO

RB-INSERT(T, z)

```
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == T.nil
10      T.root = z
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

TREE-INSERT(T, z)

```
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == NIL
10      T.root = z          // tree T was empty
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
```

```
RB-INSERT-FIXUP(T, z)
 1  while z.p.color == RED
 2      if z.p == z.p.p.left
 3          y = z.p.p.right
 4          if y.color == RED
 5              z.p.color = BLACK                    // case 1
 6              y.color = BLACK                      // case 1
 7              z.p.p.color = RED                    // case 1
 8              z = z.p.p                            // case 1
 9          else if z == z.p.right
10              z = z.p                              // case 2
11              LEFT-ROTATE(T, z)                    // case 2
12              z.p.color = BLACK                    // case 3
13              z.p.p.color = RED                    // case 3
14              RIGHT-ROTATE(T, z.p.p)               // case 3
15      else (same as then clause
              with "right" and "left" exchanged)
16  T.root.color = BLACK
```

```
RB-INSERT-FIXUP(T, z)
 1  while z.p.color == RED
 2      if z.p == z.p.p.left
 3          y = z.p.p.right
 4          if y.color == RED
 5              z.p.color = BLACK                    // case 1
 6              y.color = BLACK                      // case 1
 7              z.p.p.color = RED                    // case 1
 8              z = z.p.p                            // case 1
 9          else if z == z.p.right
10                  z = z.p                          // case 2
11                  LEFT-ROTATE(T, z)                // case 2
12              z.p.color = BLACK                    // case 3
13              z.p.p.color = RED                    // case 3
14              RIGHT-ROTATE(T, z.p.p)               // case 3
15      else (same as then clause
                with "right" and "left" exchanged)
16  T.root.color = BLACK
```

- SE T E' VUOTO, z E' LA RADICE E DUNQUE IL SUO COLORE DEVE ESSERE NERO

RB-INSERT-FIXUP(T, z)

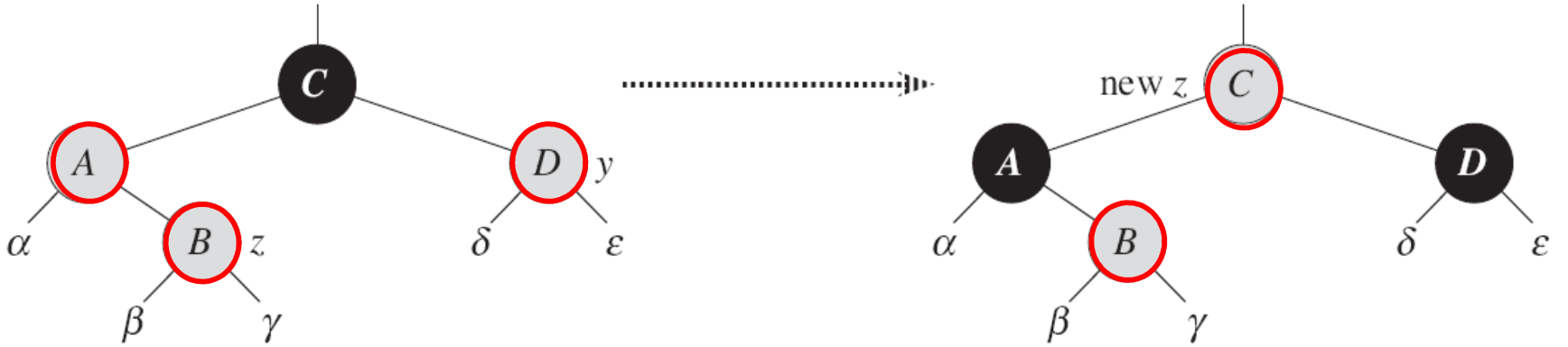
1 **while** $z.p.color$ == RED // SIN QUANDO E' VIOLATA LA CONDIZIONE
// CHE VIETA NODI ROSSI CONSECUTIVI ...

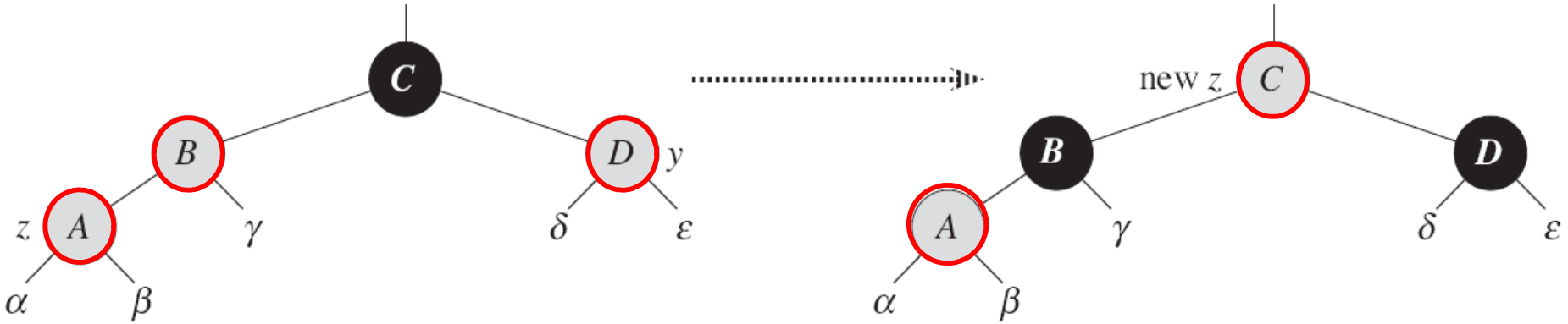
2 **if** $z.p == z.p.p.left$ // CASO z.p FIGLIO SINISTRO
// z.p NON PUÒ ESSERE LA RADICE!

3 $y = z.p.p.right$ // y E' LO ZIO DI z

CASO: ZIO DI Z "ROSSO"

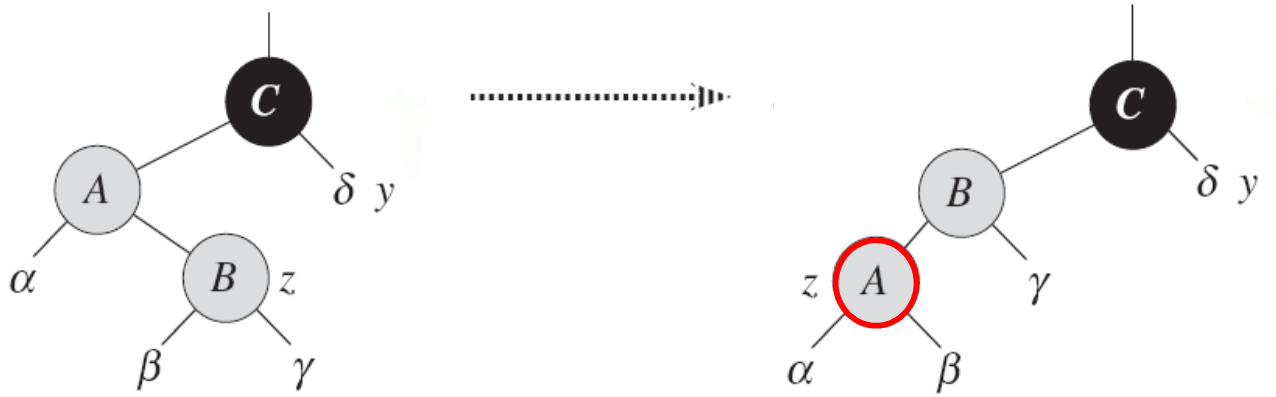
```
4        if y.color == RED
5            z.p.color = BLACK
6            y.color = BLACK
7            z.p.p.color = RED
8            z = z.p.p
```





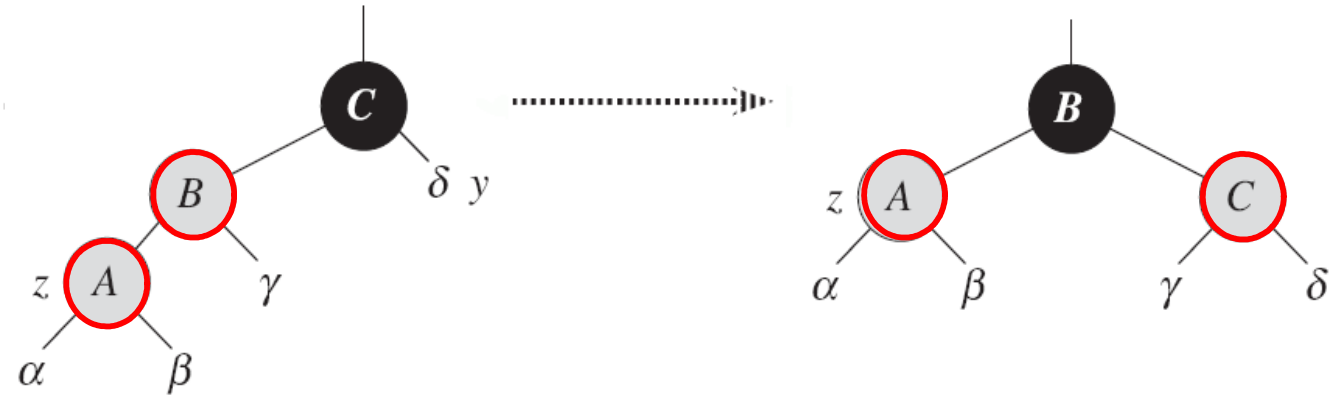
```
9     else if z == z.p.right
10          z = z.p
11          LEFT-ROTATE(T, z)
```

CASO: ZIO DI Z "NERO"
E Z E' UN FIGLIO DESTRO




```
12          z.p.color = BLACK
13          z.p.p.color = RED
14          RIGHT-ROTATE(T, z.p.p)
```

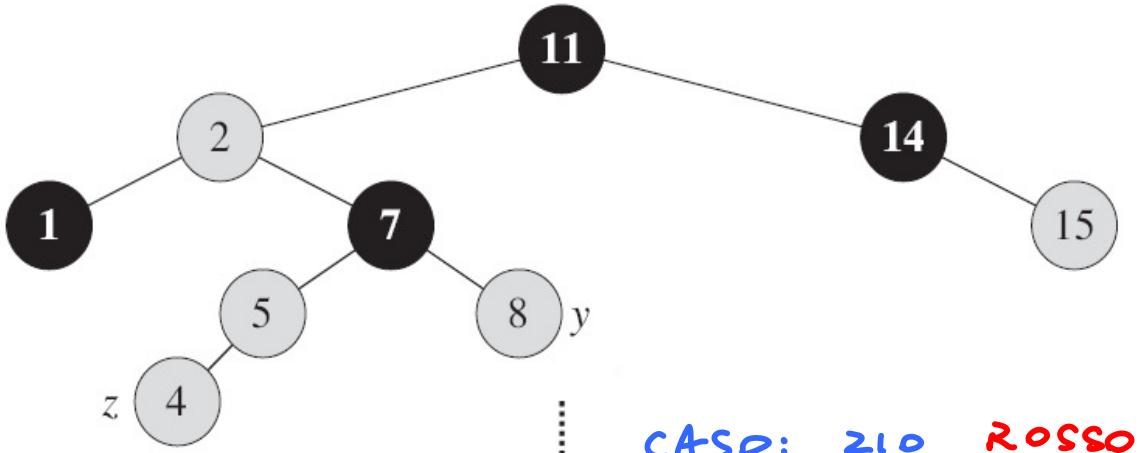
CASO: ZIO DI Z "NERO"
E Z E' UN FIGLIO SINISTRO



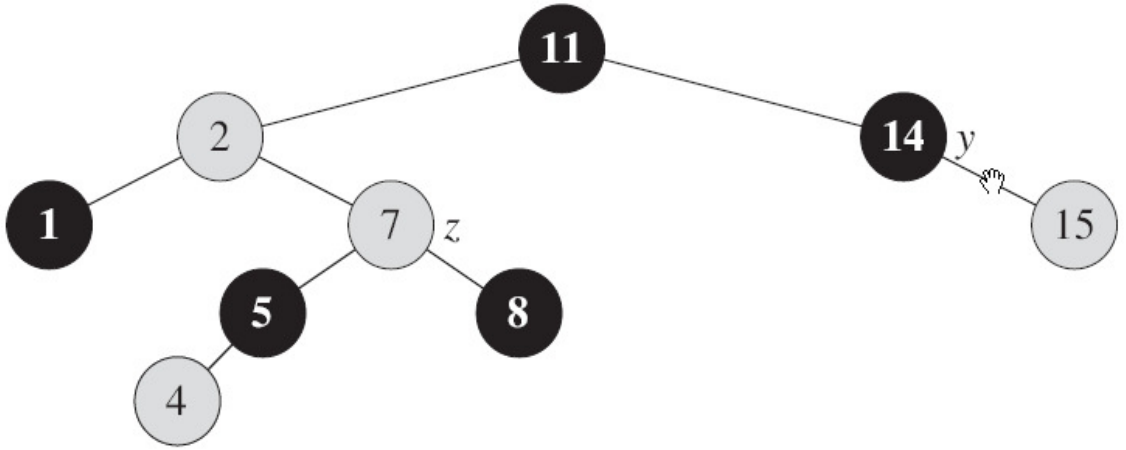
```
2       if z.p == z.p.p.left
⋮           ⋮
15      else (same as then clause with "right" and "left" exchanged)
16  T.root.color = BLACK
```

COMPLESSITA' : $O(\lg n)$

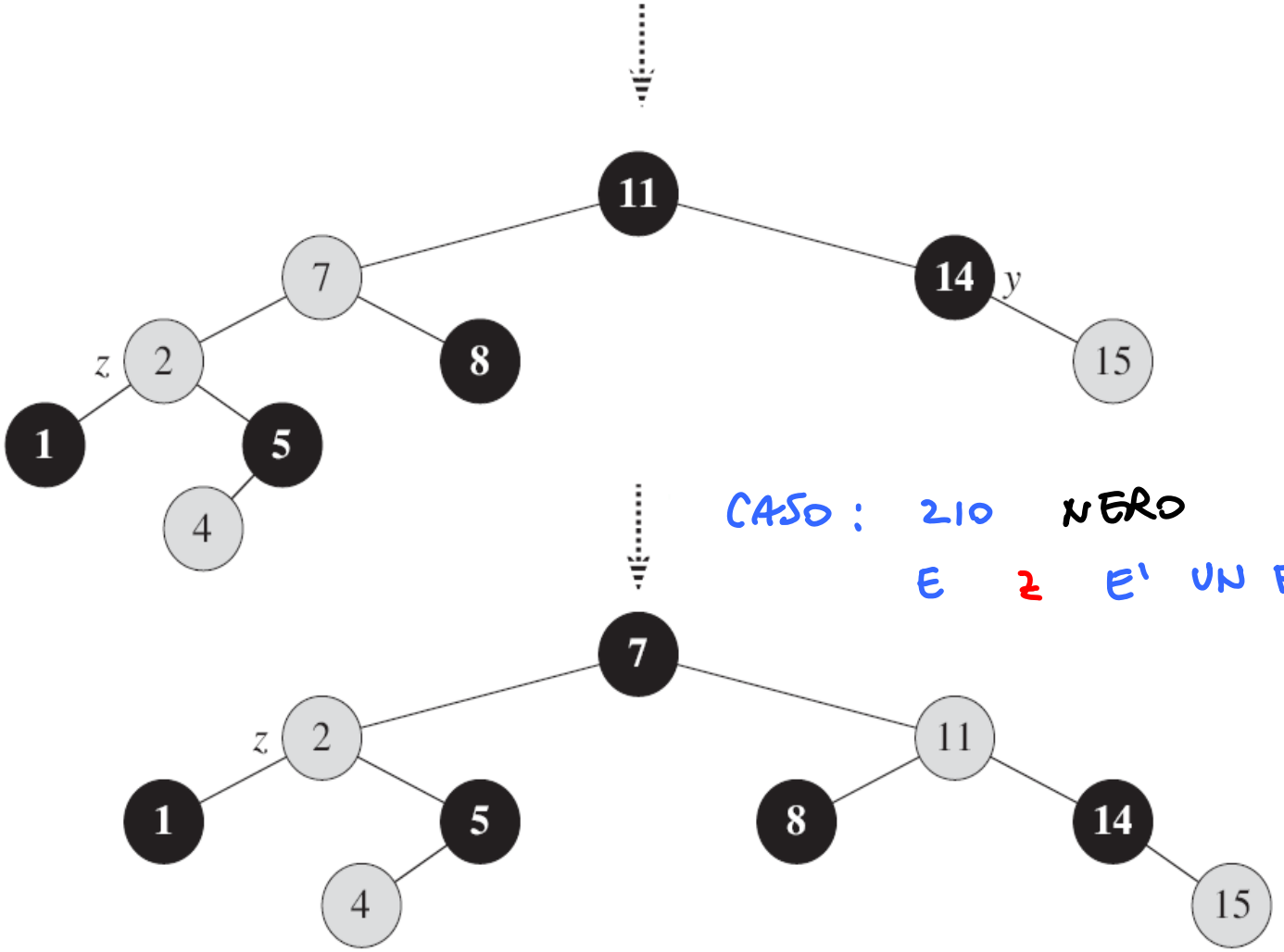
ESEMPIO (INSERIMENTO DI z CON z.key = 4)



CASO: ZIO ROSSO



CASO: ZIO NERO
E z E' UN FIGLIO DESTRO



CASO : 210 NERO

E 2 E' UN FIGLIO SINISTRO

ESERCIZI

13.3-1

In line 16 of RB-INSERT, we set the color of the newly inserted node z to red. Observe that if we had chosen to set z's color to black, then property 4 of a red-black tree would not be violated. Why didn't we choose to set z's color to black?

13.3-2

Show the red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty red-black tree.

13.3-3

Suppose that the black-height of each of the subtrees $\alpha, \beta, \gamma, \delta, \varepsilon$ in Figures 13.5 and 13.6 is k. Label each node in each figure with its black-height to verify that the indicated transformation preserves property 5.

ESERCIZI (CNTD)

13.3-4

Professor Teach is concerned that RB-INSERT-FIXUP might set $T.nil.color$ to RED, in which case the test in line 1 would not cause the loop to terminate when z is the root. Show that the professor's concern is unfounded by arguing that RB-INSERT-FIXUP never sets $T.nil.color$ to RED.

13.3-5

Consider a red-black tree formed by inserting n nodes with RB-INSERT. Argue that if $n > 1$, the tree has at least one red node.

CANCELLAZIONE

- LA PROCEDURA PER CANCELLARE UN NODO DA UN ALBERO ROSSO-NERO SI BASA SULLA PROCEDURA TREE-DELETE (E QUINDI SU TRANSPLANT)

```
RB-TRANSPLANT(T, u, v)
1  if u.p == T.nil
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
6  v.p = u.p
```

```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
6  if v ≠ NIL
7      v.p = u.p
```

- 2 DIFFERENZE ...

DIFFERENZA 1: T.nil AL POSTO DI NIL

```
RB-TRANSPLANT(T, u, v)
1  if u.p == T.nil
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
6  v.p = u.p
```

```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
6  if v ≠ NIL
7      v.p = u.p
```

DIFFERENZA 2 : L'ASSEGNAMENTO $v.p = u.p$ PUO' ESSERE FATTO ANCHE QUANDO $v == T.nil$

RB-TRANSPLANT(T, u, v)

```
1  if u.p == T.nil
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
6  v.p = u.p
```

TRANSPLANT(T, u, v)

```
1  if u.p == NIL
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
6  if v ≠ NIL
7      v.p = u.p
```

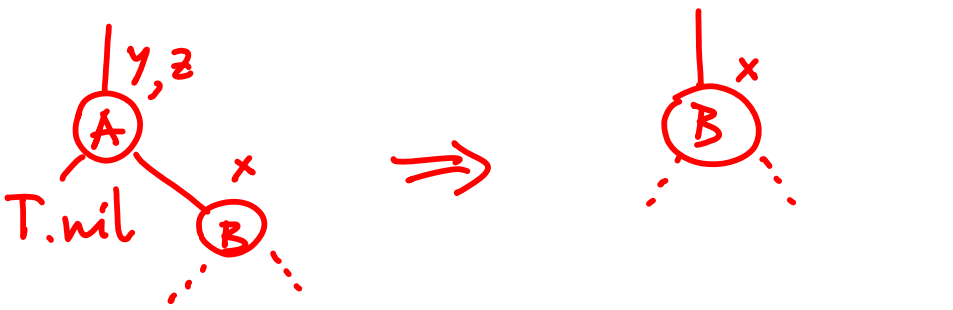
```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

- SIMILE A TREE-DELETE(T, z)

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4      x = z.right
 5      RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7      x = z.left
 8      RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10      y-original-color = y.color
11      x = y.right
12      if y.p == z
13          x.p = y
14      else RB-TRANSPLANT(T, y, y.right)
15          y.right = z.right
16          y.right.p = y
17      RB-TRANSPLANT(T, z, y)
18      y.left = z.left
19      y.left.p = y
20      y.color = z.color
21  if y-original-color == BLACK
22      RB-DELETE-FIXUP(T, x)
```

```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2      TRANSPLANT(T, z, z.right)
 3  elseif z.right == NIL
 4      TRANSPLANT(T, z, z.left)
 5  else y = TREE-MINIMUM(z.right)
 6      if y.p ≠ z
 7          TRANSPLANT(T, y, y.right)
 8          y.right = z.right
 9          y.right.p = y
10      TRANSPLANT(T, z, y)
11      y.left = z.left
12      y.left.p = y
```

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

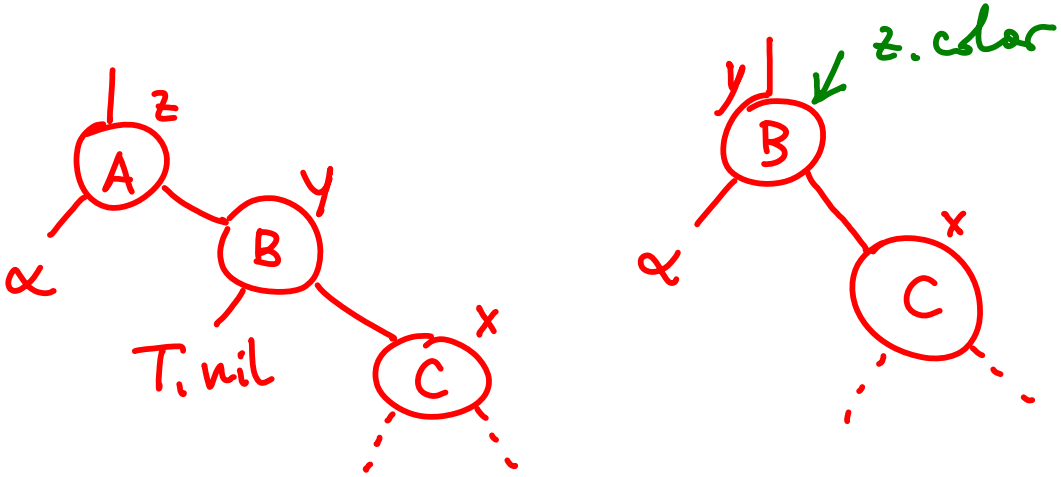


NOTA: IL NODO B POTREBBE ESSERE T.nil

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

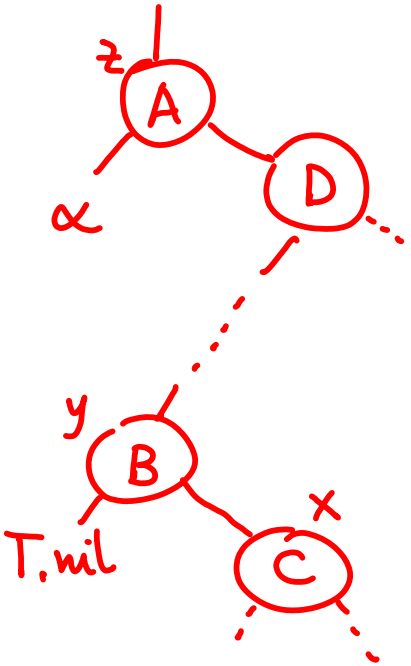
CASO: z.right.left == T.nil



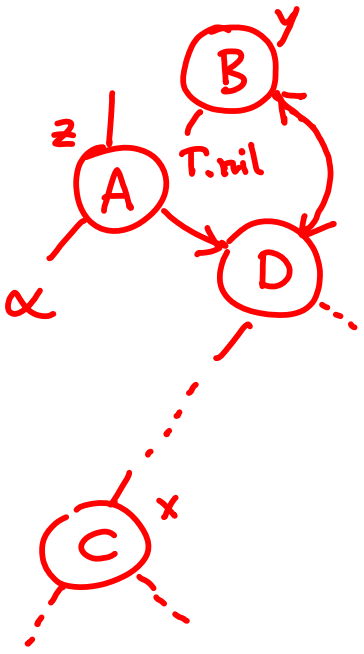
NOTA: 12-13 RILEVANTI SOLO SE x == T.nil

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

CASO: z.right.left ≠ T.nil

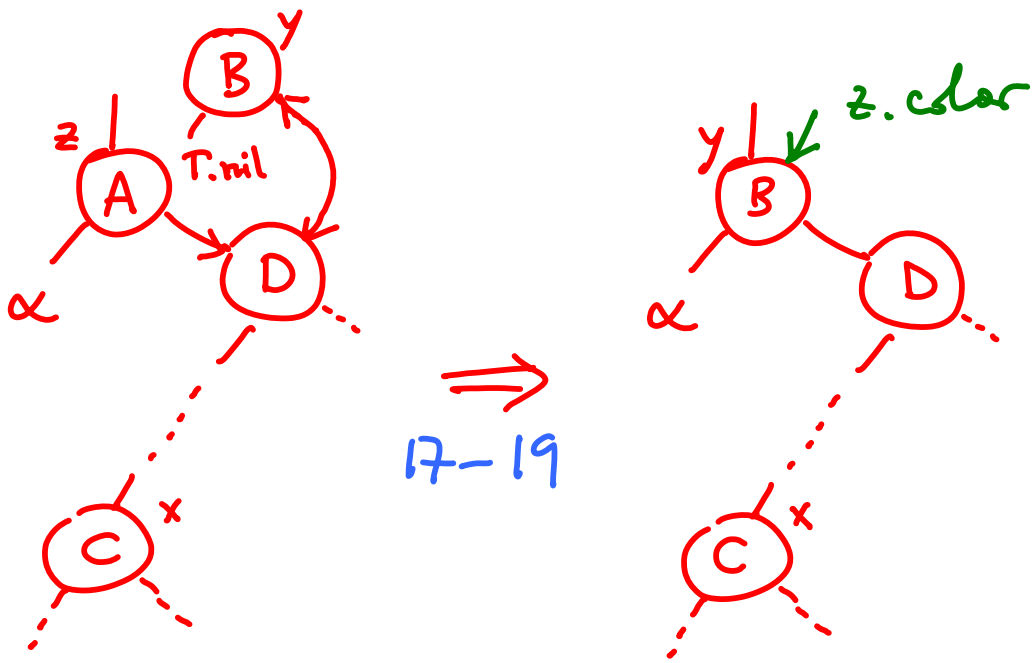


⇒ 14-16



```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y.p == z
13            x.p = y
14       else RB-TRANSPLANT(T, y, y.right)
15            y.right = z.right
16            y.right.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

CASO: z.right.left ≠ T.nil



- SE y-original-color == BLACK, IN TUTTI I CAMMINI SEMPLICI DA T.root AD UNA FOGLIA PASSANTI PER x VIENE A MANCARE UN NODO NERO

- LA PROCEDURA RB-DELETE-FIXUP (T, x) RIPRISTINERA' LA PROPRIETA' SULLE ALTEZZE NERE

- POSSIAMO IMMAGINARE CHE SE y-original-color == BLACK, IL NODO x RICEVA UNA DOSE EXTRA DI NERO:

  ROSSO + NERO = NERO

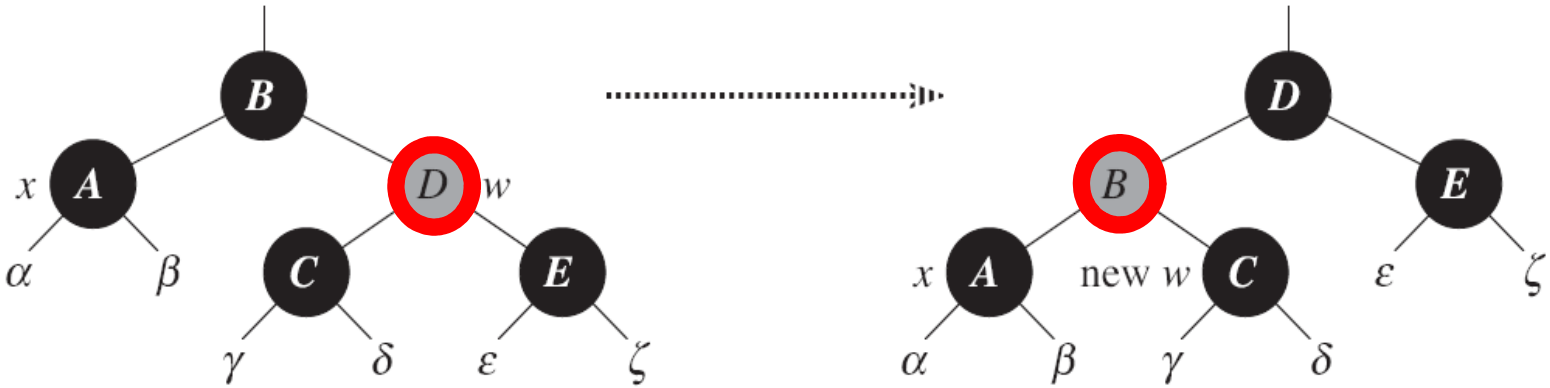
  NERO + NERO = DOPPIO NERO

- LA PROCEDURA RB-DELETE-FIXUP (T, x) GESTIRA' OPPORTUNAMENTE L'EVENTUALE DOPPIO NERO

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 4          if w.color == RED
 5              w.color = BLACK
 6              x.p.color = RED
 7              LEFT-ROTATE(T, x.p)
 8              w = x.p.right
 9          if w.left.color == BLACK and w.right.color == BLACK
10              w.color = RED
11              x = x.p
12          else if w.right.color == BLACK
13                  w.left.color = BLACK
14                  w.color = RED
15                  RIGHT-ROTATE(T, w)
16                  w = x.p.right
17              w.color = x.p.color
18              x.p.color = BLACK
19              w.right.color = BLACK
20              LEFT-ROTATE(T, x.p)
21              x = T.root
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```

CASO: FRATELLO ROSSO

```
RB-DELETE-FIXUP(T, x)
1  while x ≠ T.root and x.color == BLACK
2      if x == x.p.left
3          w = x.p.right
4          if w.color == RED
5              w.color = BLACK
6              x.p.color = RED
7              LEFT-ROTATE(T, x.p)
8              w = x.p.right
```

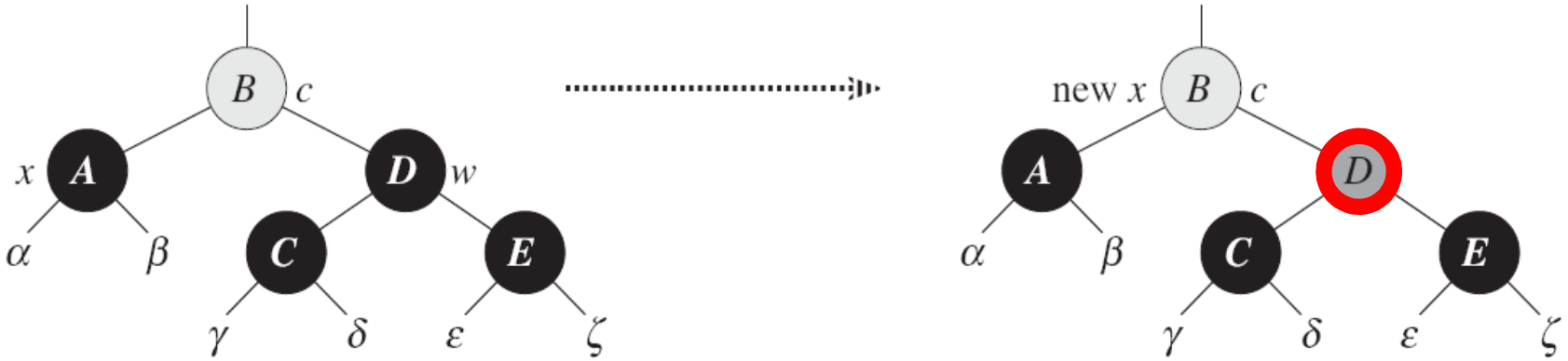


CASO : - FRATELLO NERO

- NIPOTI NERI

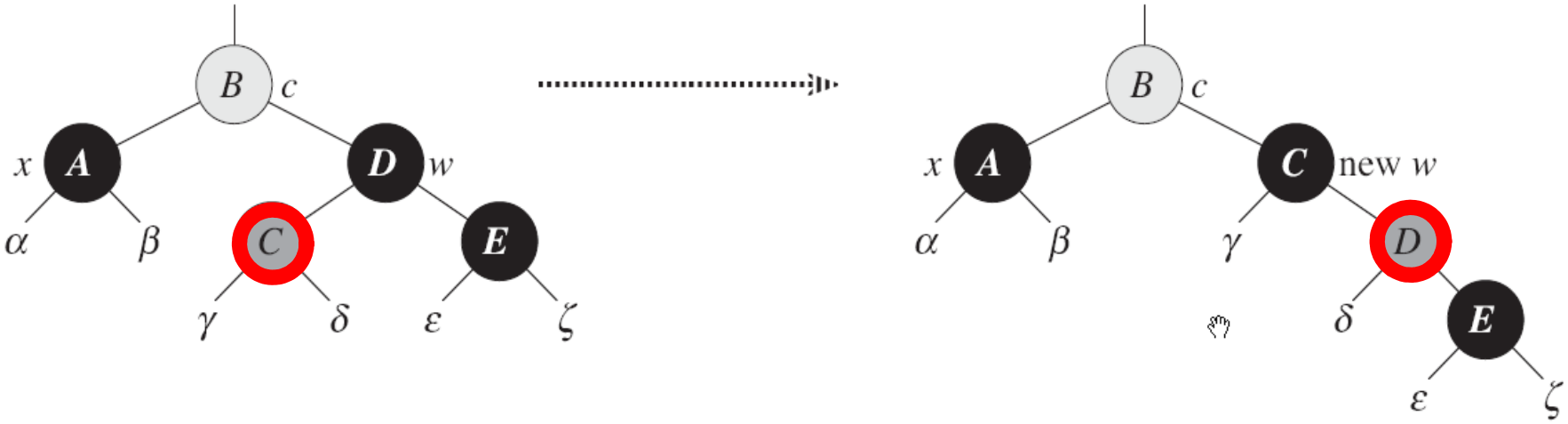
```
4        if w.color == RED
5            w.color = BLACK
6            x.p.color = RED
7            LEFT-ROTATE(T, x.p)
8            w = x.p.right
9        if w.left.color == BLACK and w.right.color == BLACK
10           w.color = RED
11           x = x.p
```

// w.color == BLACK



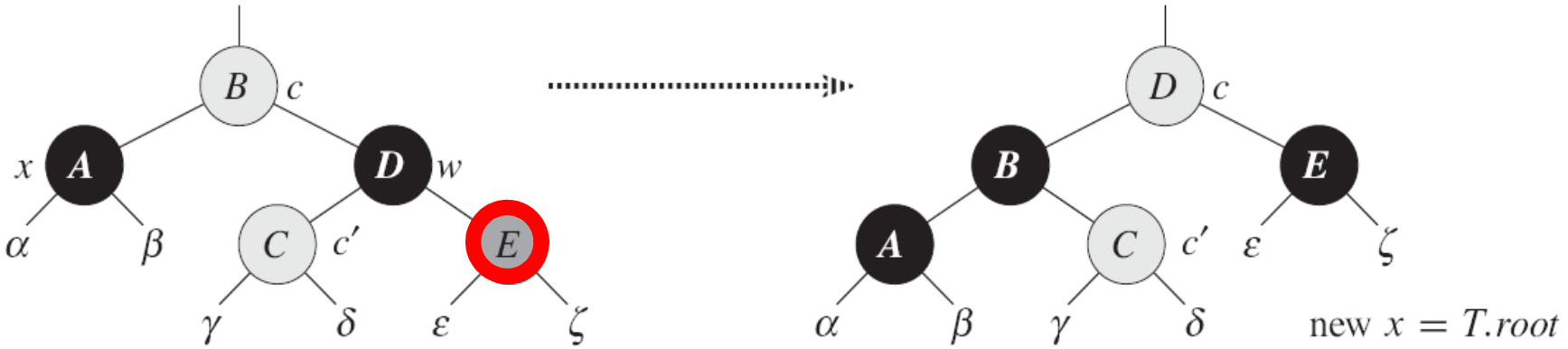
CASO : - FRATELLO NERO
- NIPOTE DESTRO NERO
- NIPOTE SINISTRO ROSSO

```
12        else if w.right.color == BLACK
13                w.left.color = BLACK
14                w.color = RED
15                RIGHT-ROTATE(T, w)
16                w = x.p.right
```



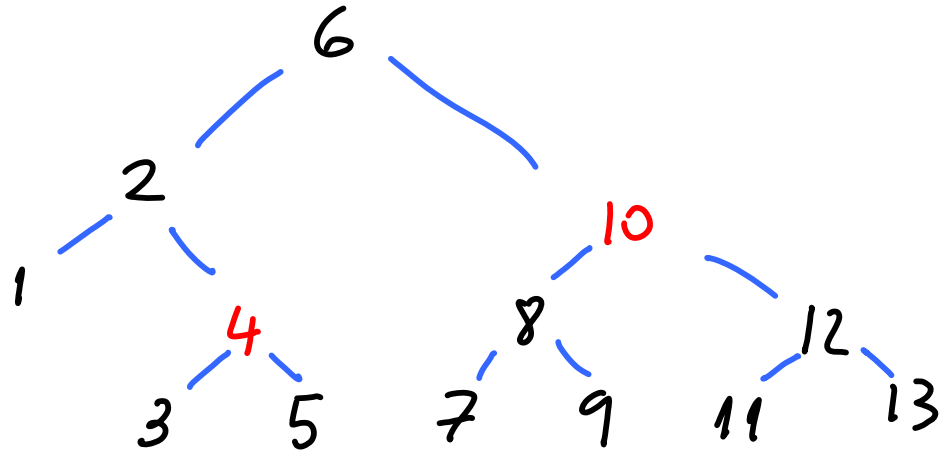
CASO CONCLUSIVO: - FRATELLO NERO
- NIPOTE DESTRO ROSSO

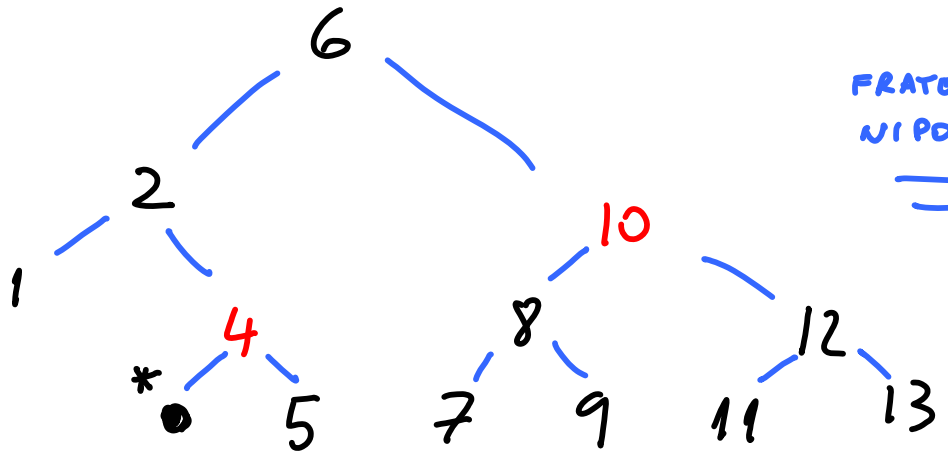
```
17    w.color = x.p.color
18    x.p.color = BLACK
19    w.right.color = BLACK
20    LEFT-ROTATE(T, x.p)
21    x = T.root
```



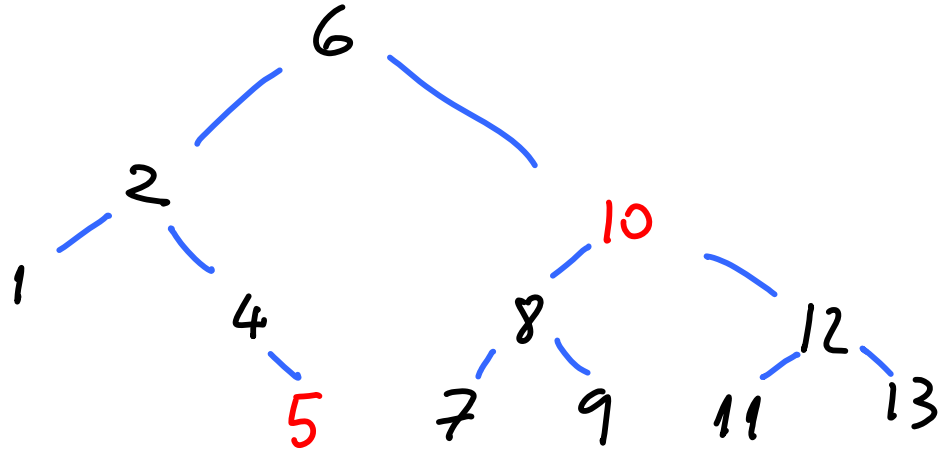
ESEMPI

RB-DELETE(T, 3)

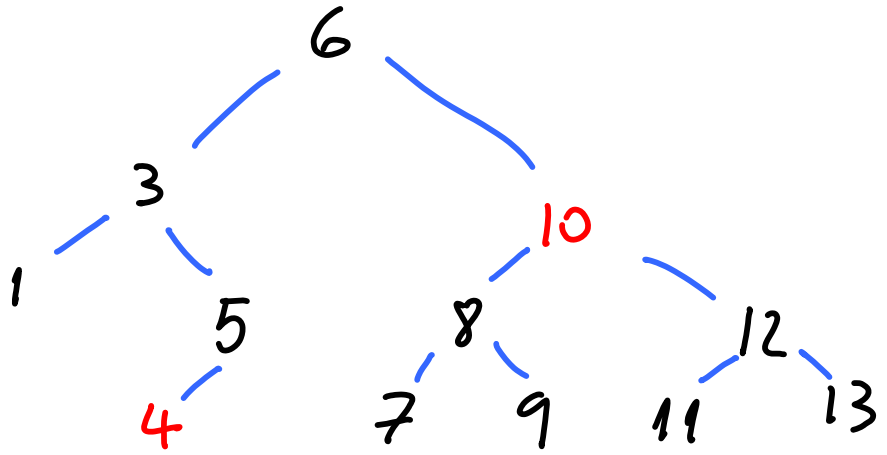


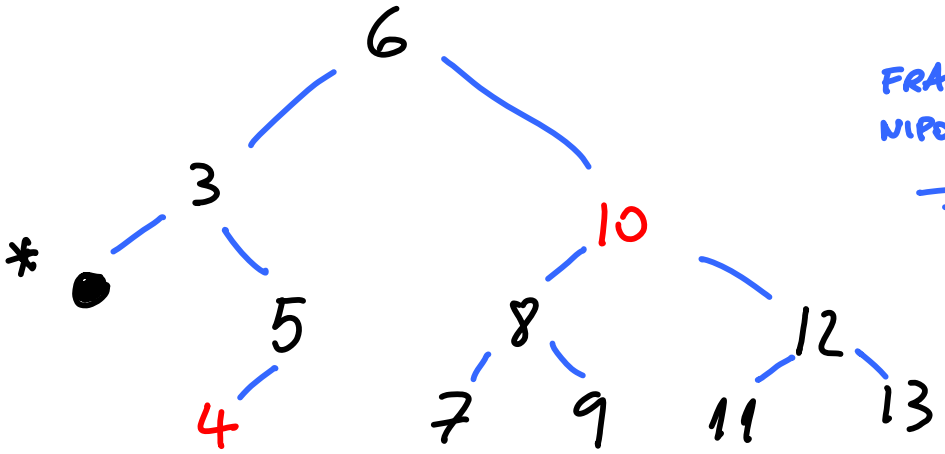


FRATELLO NERO
NIPOTI NERI

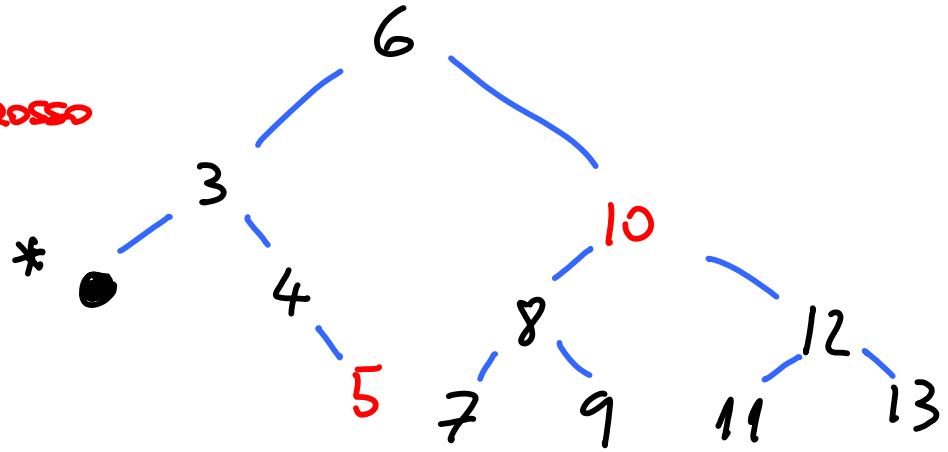


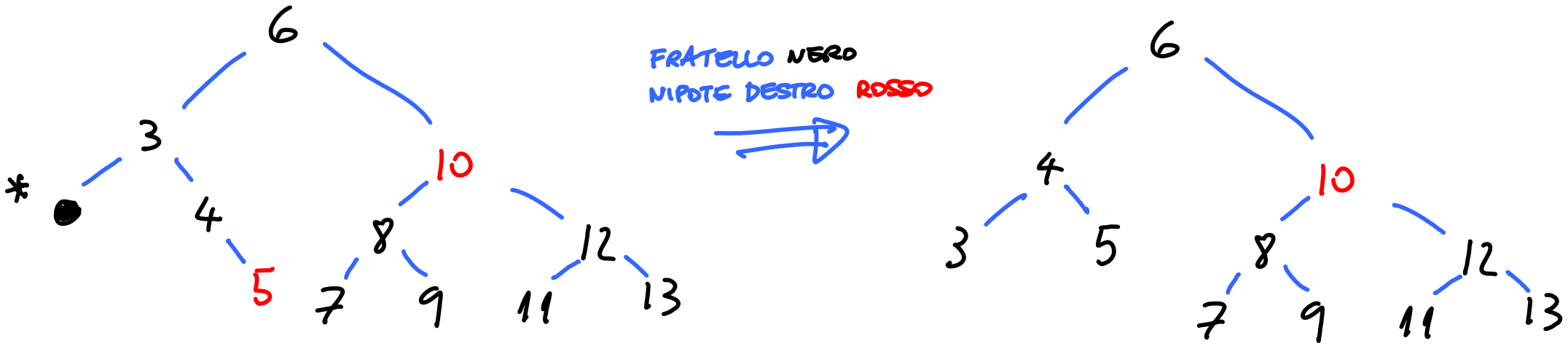
RB-DELETE(T, 1)

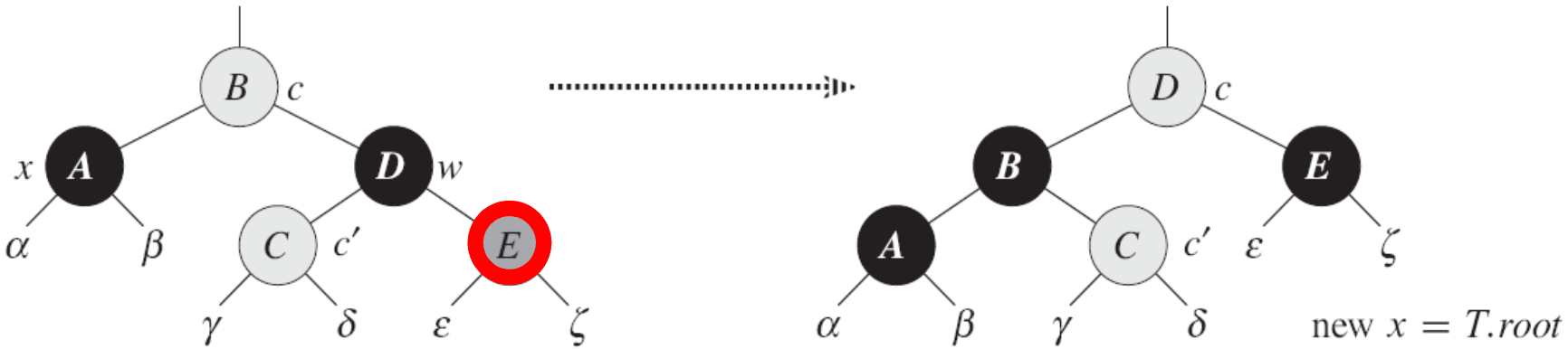




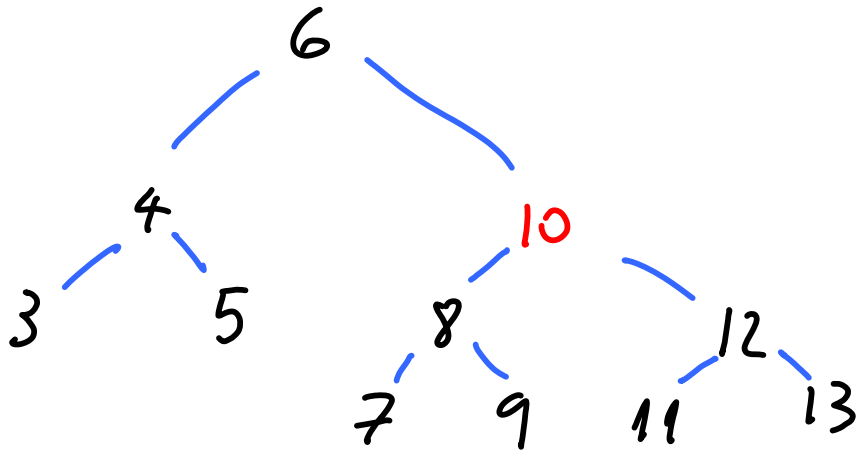
FRATELLO NERO
NIPOTE SINISTRO ROSSO

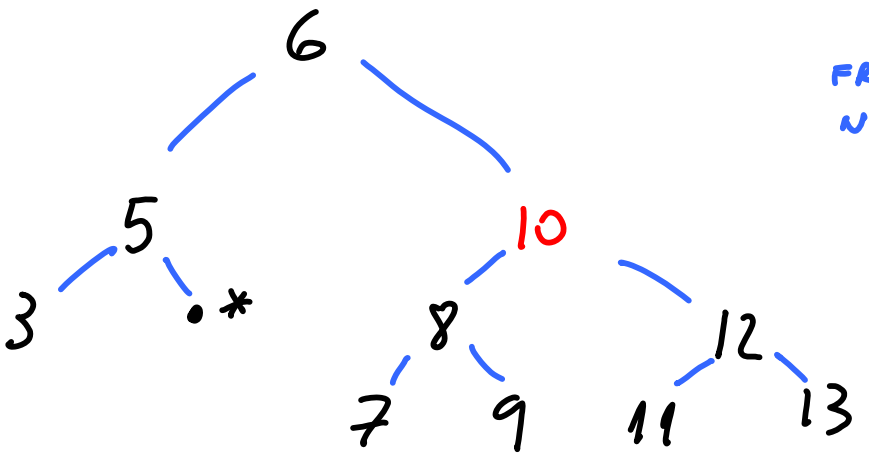



6
3
10
*
4
8
12
5
7
9
11
13
FRATELLO NERO
NIPOTE DESTRO ROSSO
6
4
10
3
5
8
12
7
9
11
13

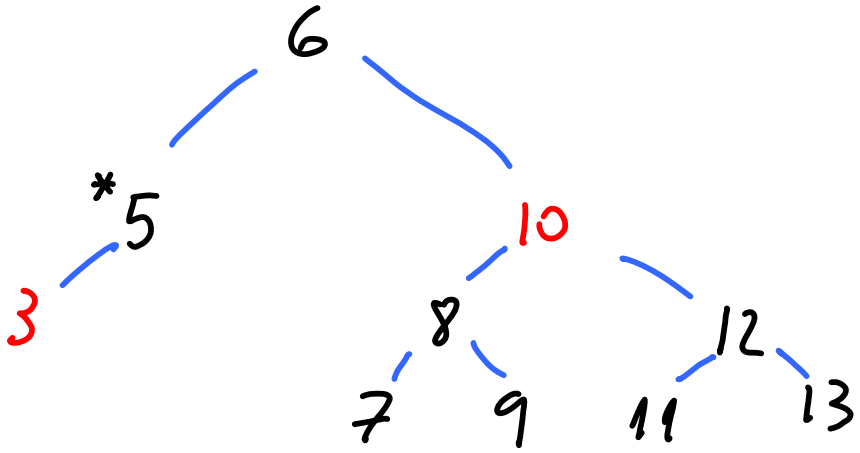

B
c
x
A
D
w
α
β
C
c′
E
γ
δ
ε
ζ
D
c
B
E
A
C
c′
ε
ζ
α
β
γ
δ
new x = T.root

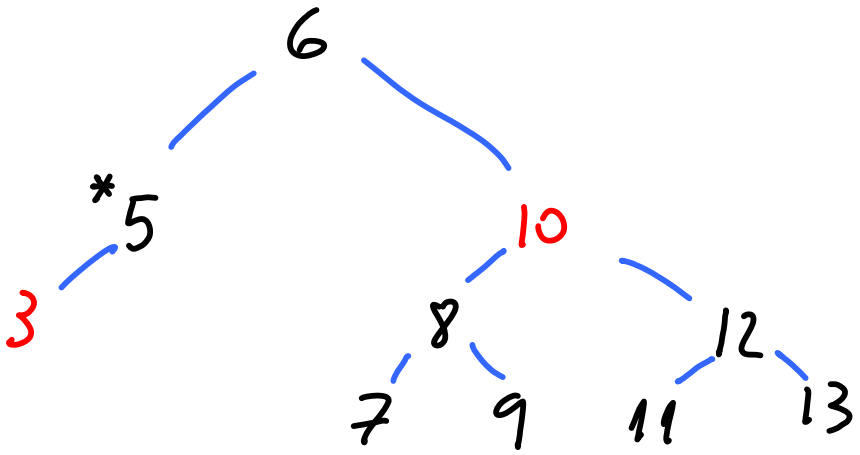
RB-DELETE(T, 4)



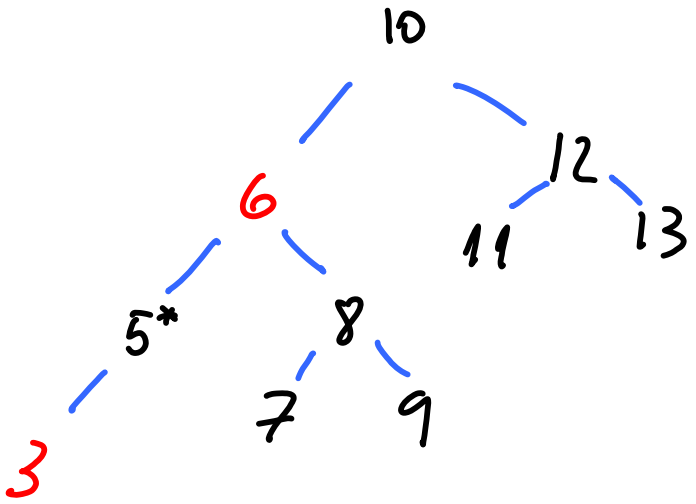


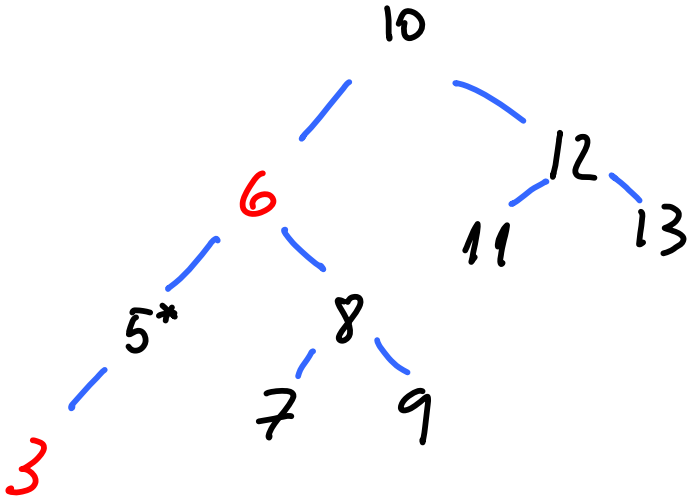
FRATELLO NERO
NIPOTI NERI



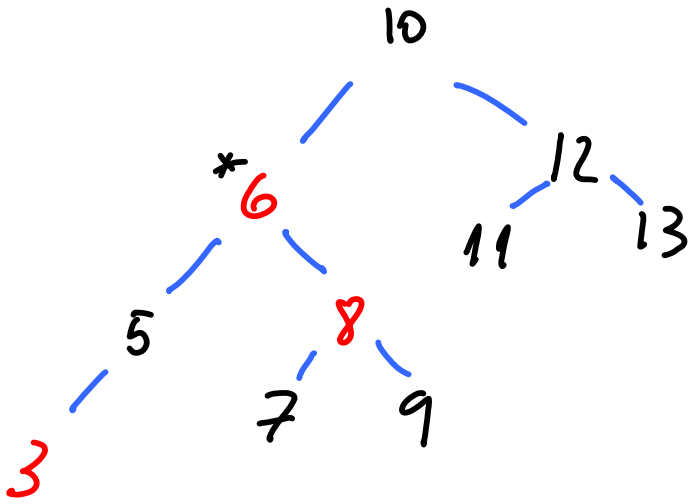


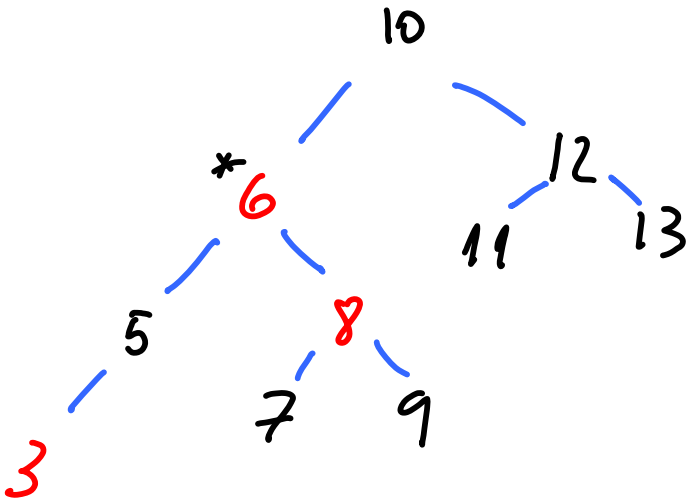
FRATELLO ROSSO



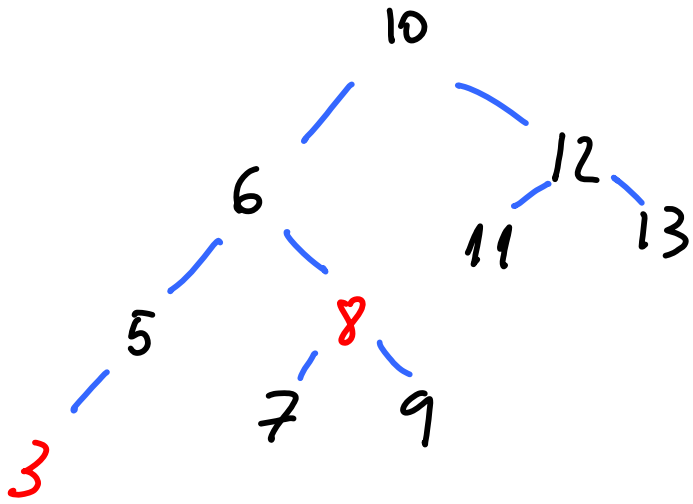


FRATELLO NERO
NIPOTI NERI





ISTRUZIONE 27



ESERCIZIO

13.4-3

In Exercise 13.3-2, you found the red-black tree that results from successively inserting the keys $41, 38, 31, 12, 19, 8$ into an initially empty tree. Now show the red-black trees that result from the successive deletion of the keys in the order $8, 12, 19, 31, 38, 41$.