
Algorithms and Data Structures

CSCI 4041

Session 13

Red-Black Insert

rbInsert(Tree T, Node z):

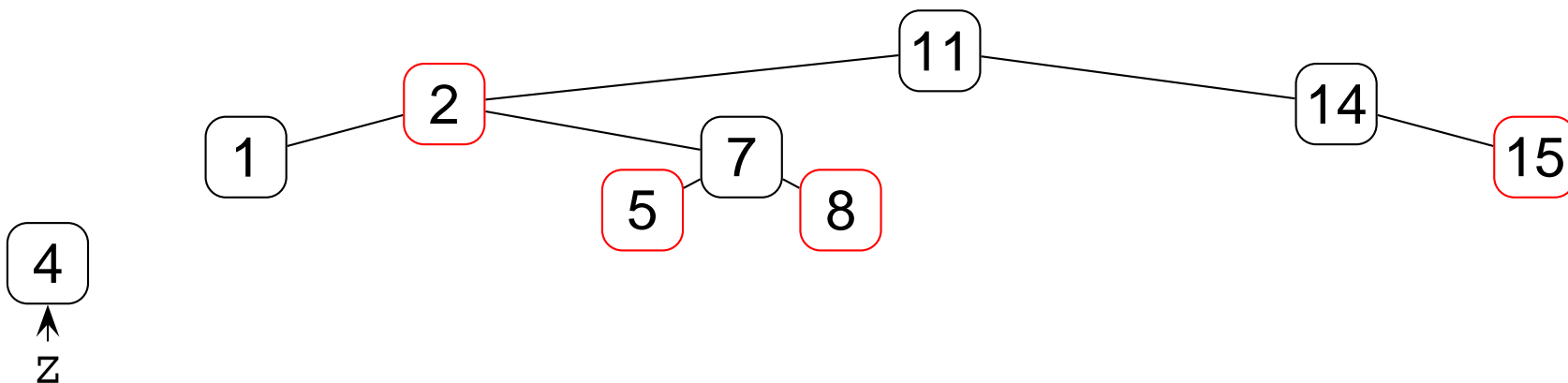
```
loop from y ← nil(T), x ← root(T) while x ≠ nil(T)  
    y ← x, if key(z) < key(x) then x ← left(x)  
                else x ← right(x)
```

p(z) ← y

```
if y = nil(T) then root(T) ← z
```

```
    else if key(z) < key(y) then left(y) ← z  
                else right(y) ← z
```

```
left(z) ← right(z) ← nil(T), col(z) ← RED, rbInsertFix(T, z)
```



Red-Black Insert (1)

rbInsert(Tree T, Node z):

loop from $y \leftarrow \text{nil}(T)$, $x \leftarrow \text{root}(T)$ **while** $x \neq \text{nil}(T)$

$y \leftarrow x$, **if** $\text{key}(z) < \text{key}(x)$ **then** $x \leftarrow \text{left}(x)$

else $x \leftarrow \text{right}(x)$

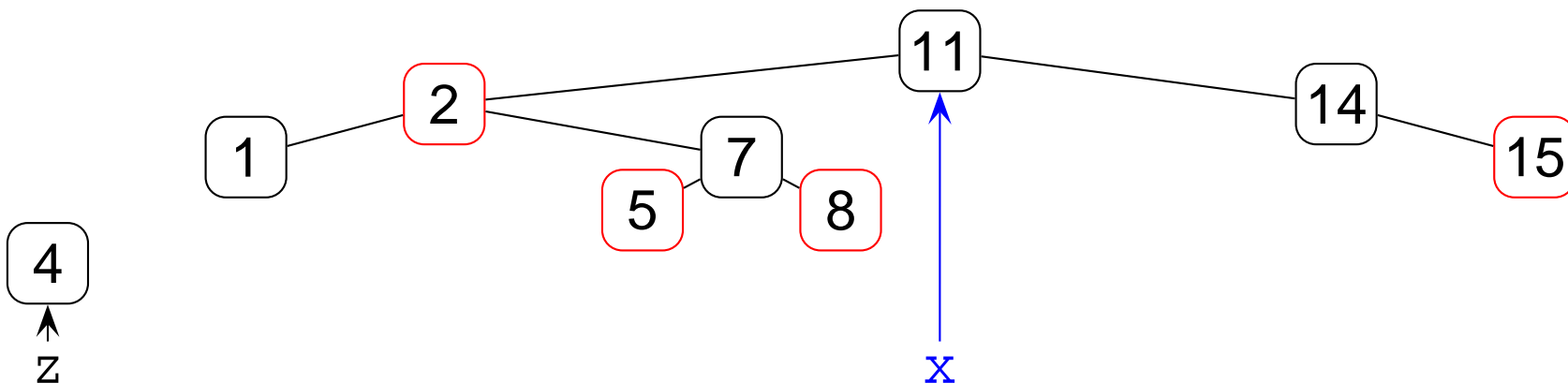
$p(z) \leftarrow y$

if $y = \text{nil}(T)$ **then** $\text{root}(T) \leftarrow z$

else if $\text{key}(z) < \text{key}(y)$ **then** $\text{left}(y) \leftarrow z$

else $\text{right}(y) \leftarrow z$

$\text{left}(z) \leftarrow \text{right}(z) \leftarrow \text{nil}(T)$, $\text{col}(z) \leftarrow \text{RED}$, $\text{rbInsertFix}(T, z)$



Red-Black Insert (2)

rbInsert(Tree T, Node z):

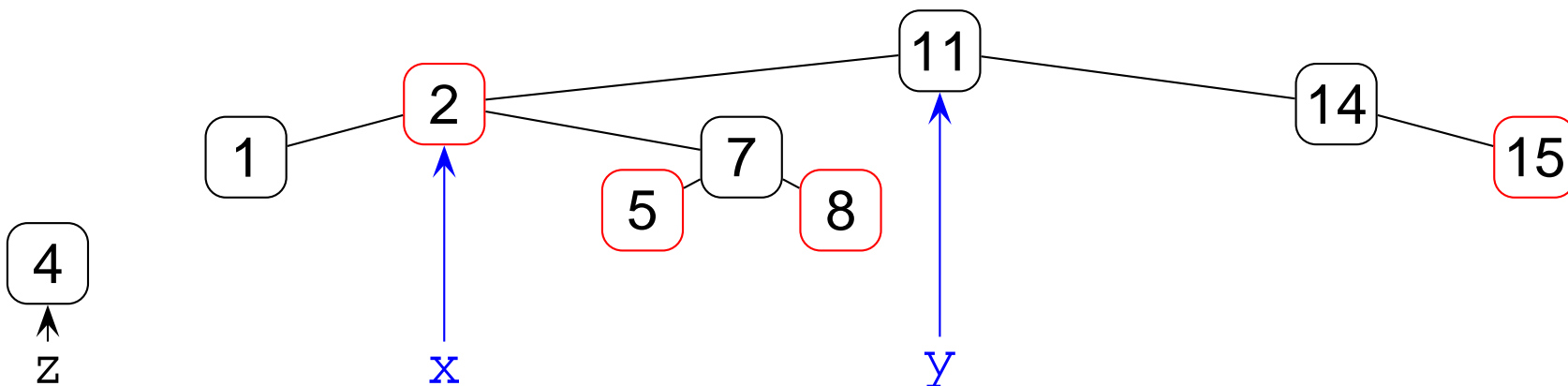
```
loop from y ← nil(T), x ← root(T) while x ≠ nil(T)  
    y ← x, if key(z) < key(x) then x ← left(x)  
    else x ← right(x)
```

p(z) ← y

```
if y = nil(T) then root(T) ← z
```

```
    else if key(z) < key(y) then left(y) ← z  
    else right(y) ← z
```

```
left(z) ← right(z) ← nil(T), col(z) ← RED, rbInsertFix(T, z)
```



Red-Black Insert (3)

rbInsert(Tree T, Node z):

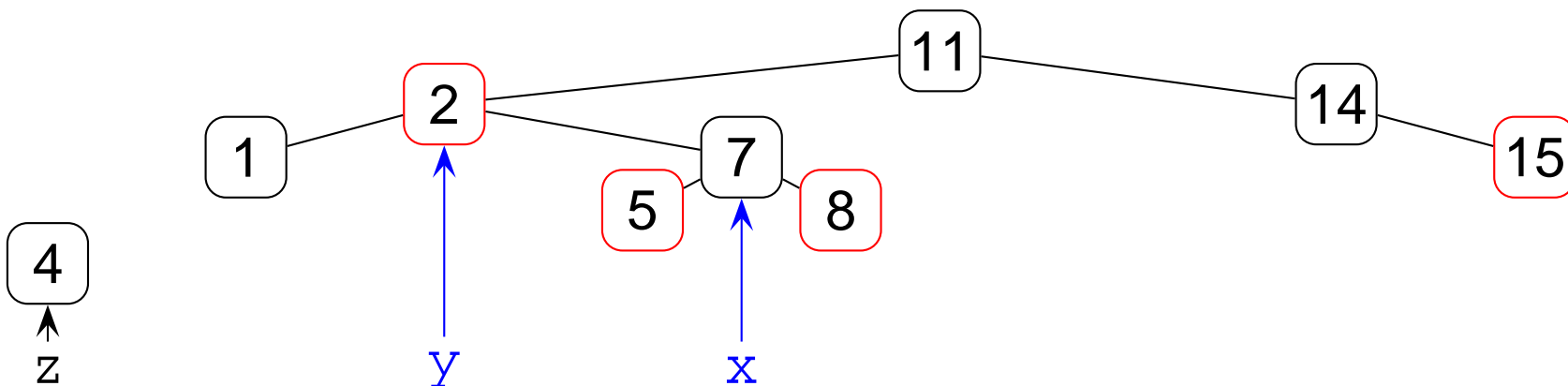
```
loop from y ← nil(T), x ← root(T) while x ≠ nil(T)  
    y ← x, if key(z) < key(x) then x ← left(x)  
    else x ← right(x)
```

p(z) ← y

```
if y = nil(T) then root(T) ← z
```

```
    else if key(z) < key(y) then left(y) ← z  
    else right(y) ← z
```

```
left(z) ← right(z) ← nil(T), col(z) ← RED, rbInsertFix(T, z)
```



Red-Black Insert (4)

rbInsert(Tree T, Node z):

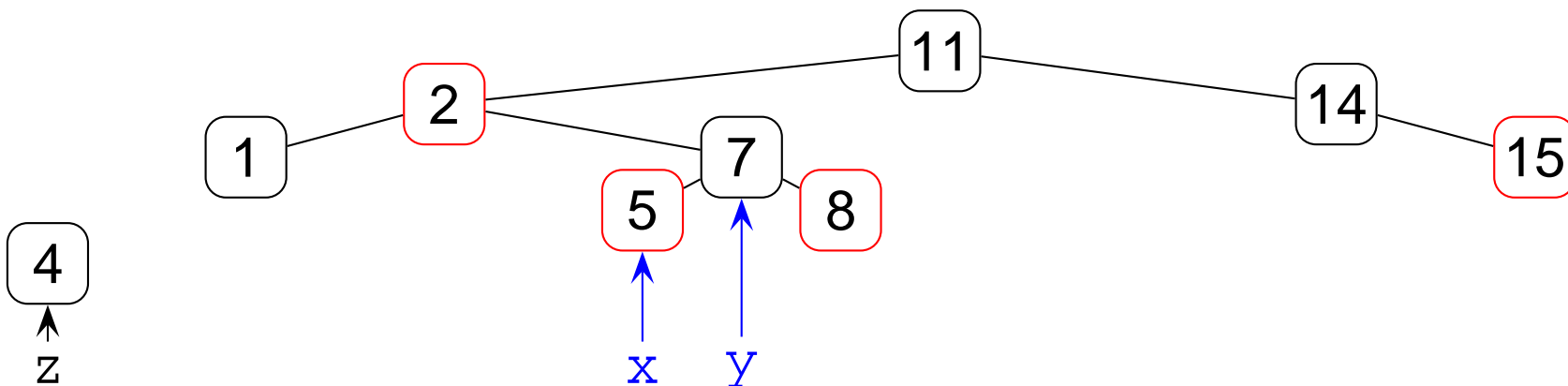
```
loop from y ← nil(T), x ← root(T) while x ≠ nil(T)  
    y ← x, if key(z) < key(x) then x ← left(x)  
    else x ← right(x)
```

p(z) ← y

```
if y = nil(T) then root(T) ← z
```

```
    else if key(z) < key(y) then left(y) ← z  
    else right(y) ← z
```

```
left(z) ← right(z) ← nil(T), col(z) ← RED, rbInsertFix(T, z)
```



Red-Black Insert (5)

rbInsert(Tree T, Node z):

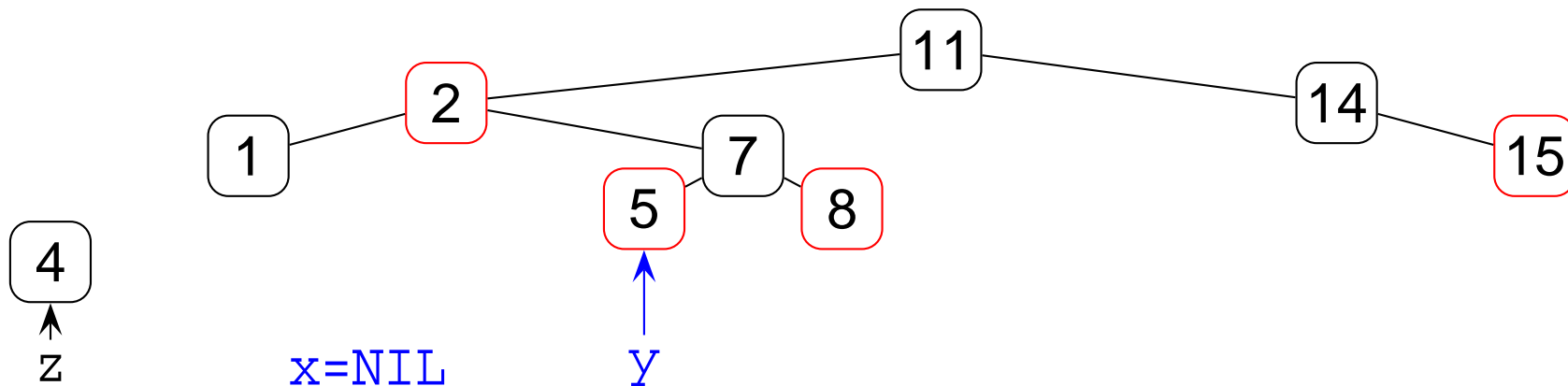
```
loop from y ← nil(T), x ← root(T) while x ≠ nil(T)  
    y ← x, if key(z) < key(x) then x ← left(x)  
    else x ← right(x)
```

p(z) ← y

```
if y = nil(T) then root(T) ← z
```

```
    else if key(z) < key(y) then left(y) ← z  
    else right(y) ← z
```

```
left(z) ← right(z) ← nil(T), col(z) ← RED, rbInsertFix(T, z)
```



Red-Black Insert (6)

rbInsert(Tree T, Node z):

```
loop from y ← nil(T), x ← root(T) while x ≠ nil(T)  
    y ← x, if key(z) < key(x) then x ← left(x)  
    else x ← right(x)
```

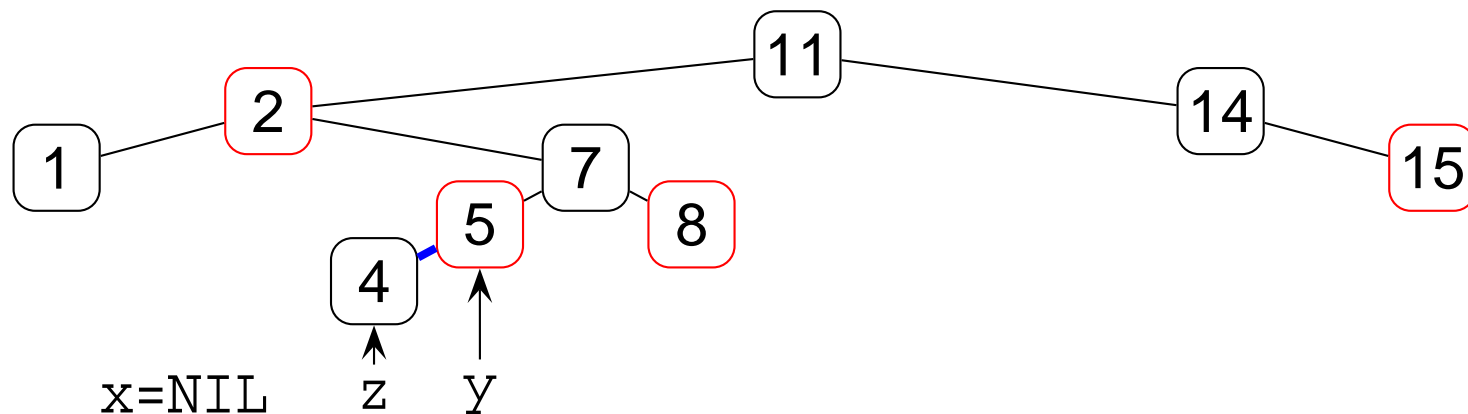
p(z) ← y

```
if y = nil(T) then root(T) ← z
```

```
    else if key(z) < key(y) then left(y) ← z
```

```
        else right(y) ← z
```

```
left(z) ← right(z) ← nil(T), col(z) ← RED, rbInsertFix(T, z)
```



Red-Black Insert (8)

rbInsert(Tree T, Node z):

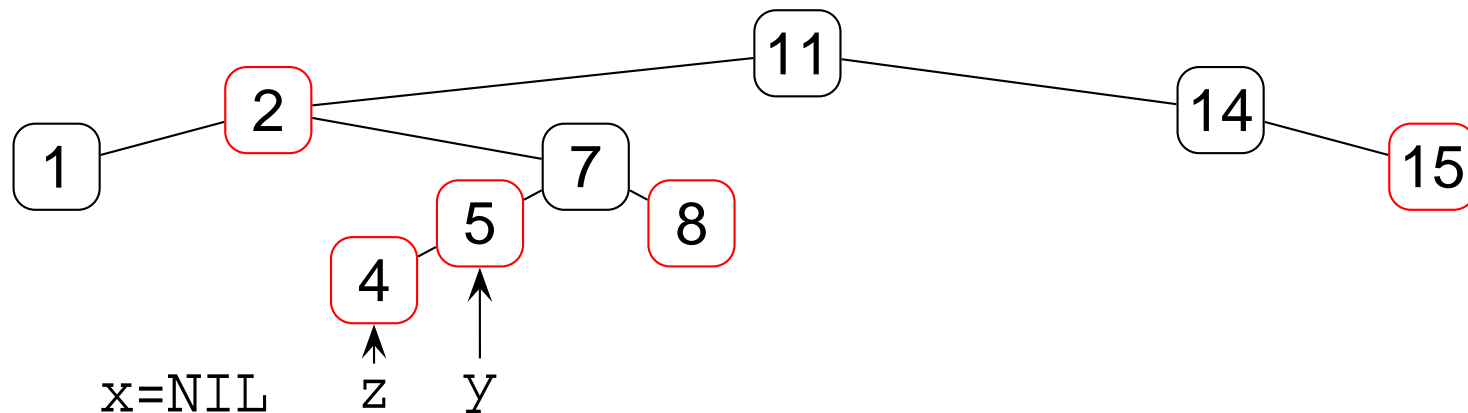
```
loop from y ← nil(T), x ← root(T) while x ≠ nil(T)  
  y ← x, if key(z) < key(x) then x ← left(x)  
  else x ← right(x)
```

p(z) ← y

```
if y = nil(T) then root(T) ← z
```

```
  else if key(z) < key(y) then left(y) ← z  
  else right(y) ← z
```

```
left(z) ← right(z) ← nil(T), col(z) ← RED, rbInsertFix(T, z)
```



Red-Black Insert (8) | Red-Black Insert Fixup

`rbInsertFix(Tree T, Node z):`

while `col(p(z))=RED`

if `p(z)=left(p(p(z)))` **then**

`y ← right(p(p(z)))`

if `col(y)=RED` **then**

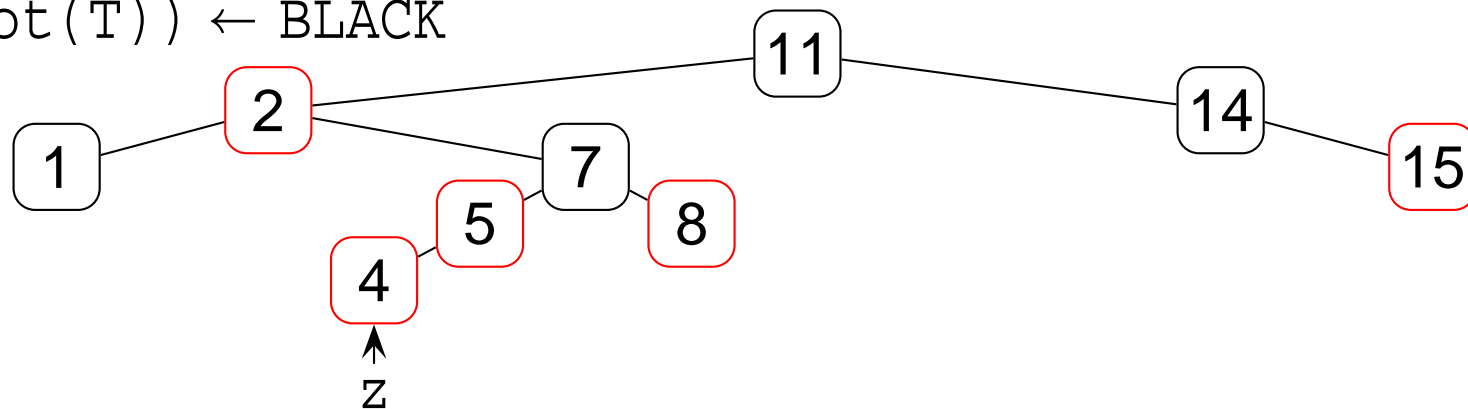
`col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))`

else if `z=right(p(z))` **then** `z ← p(z), lRot(T,z)`

`col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))`

else ... (same as **then** clause, with 'right' and 'left' exchanged)

`col(root(T)) ← BLACK`



Red-Black Insert (8) | Red-Black Insert Fixup (1)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

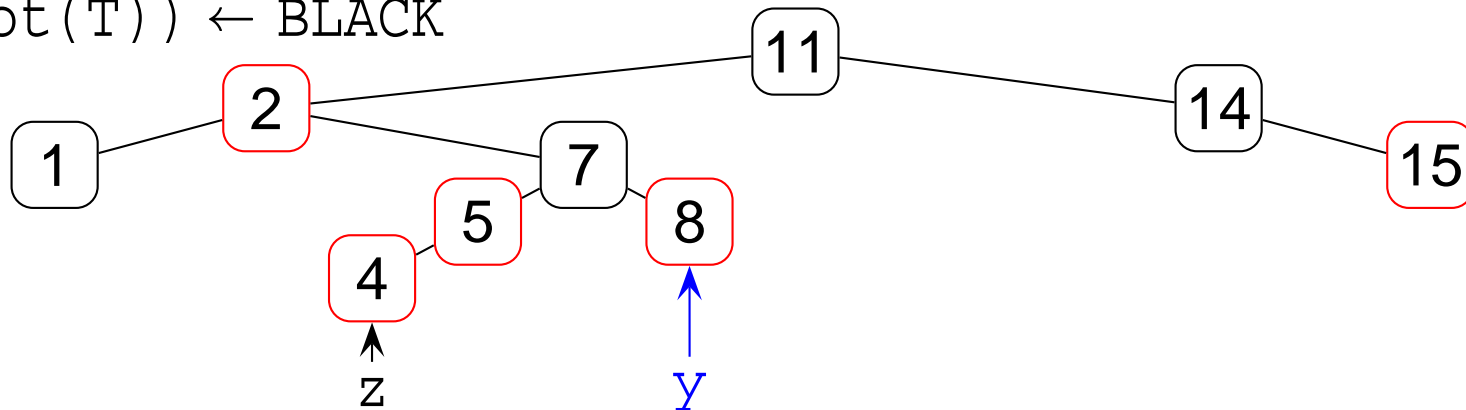
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (2)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

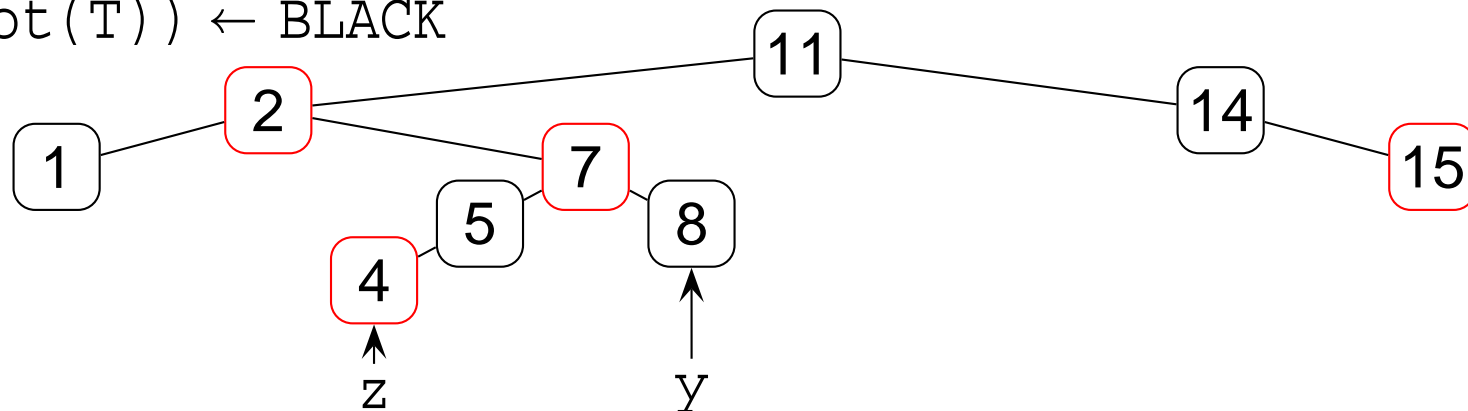
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (3)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

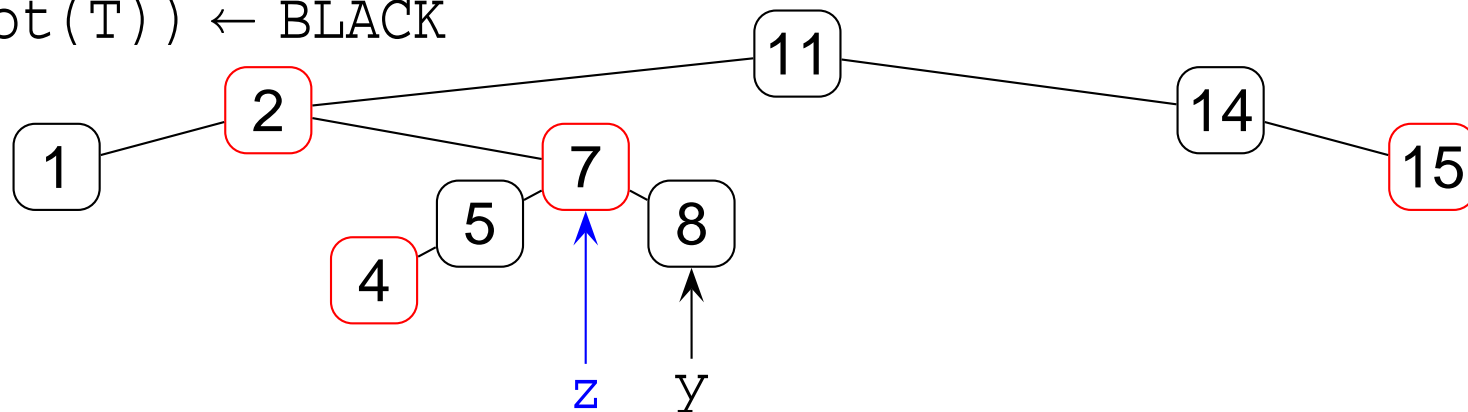
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (4)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

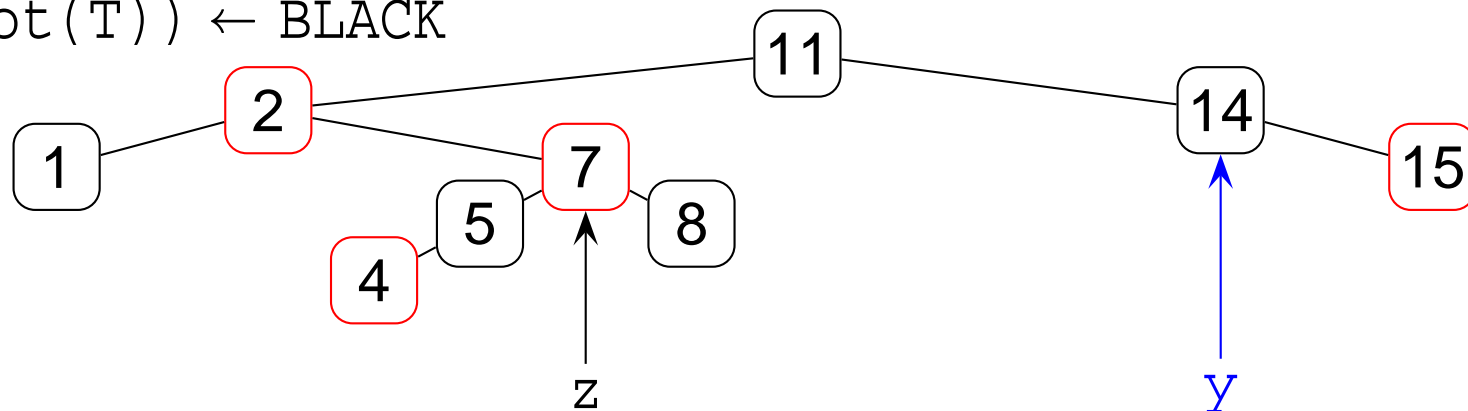
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (5)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

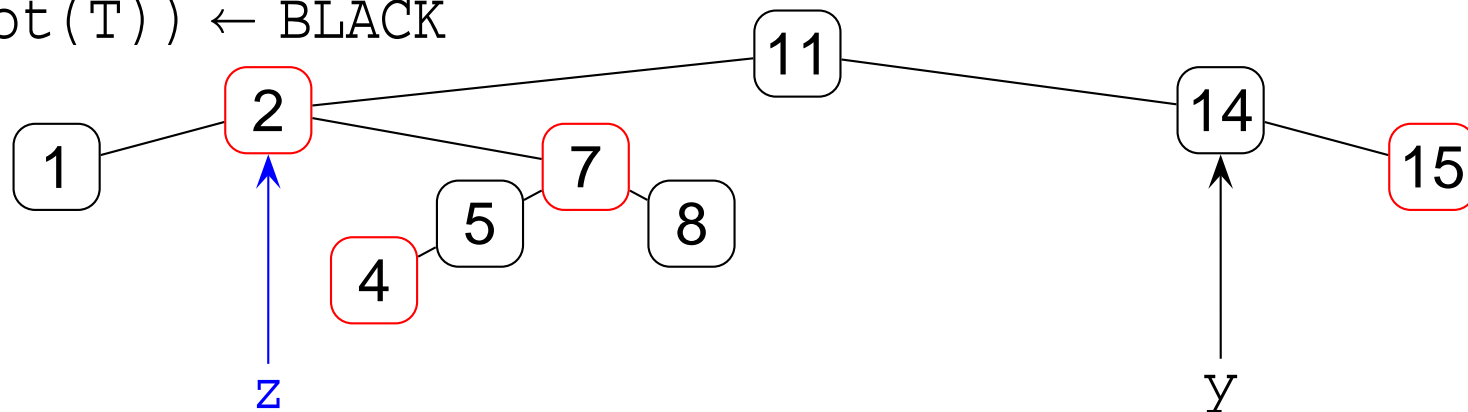
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (6)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

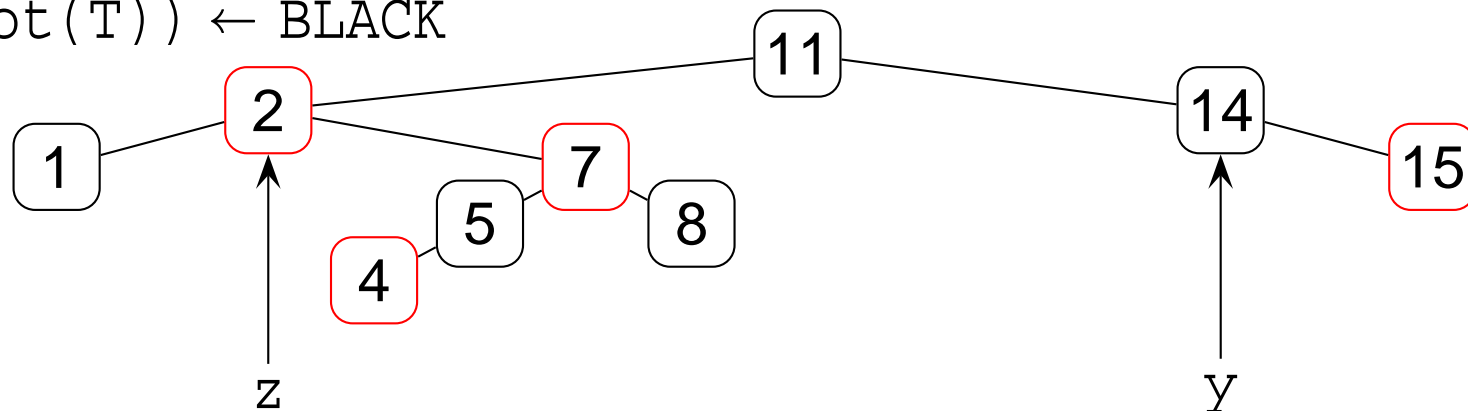
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (6) | Left Rotate

`lRot(Tree T, Node x):`

`y ← right(x)`

`right(x) ← left(y)`

if `left(y) ≠ nil(T)` **then** `p(left(y)) ← x`

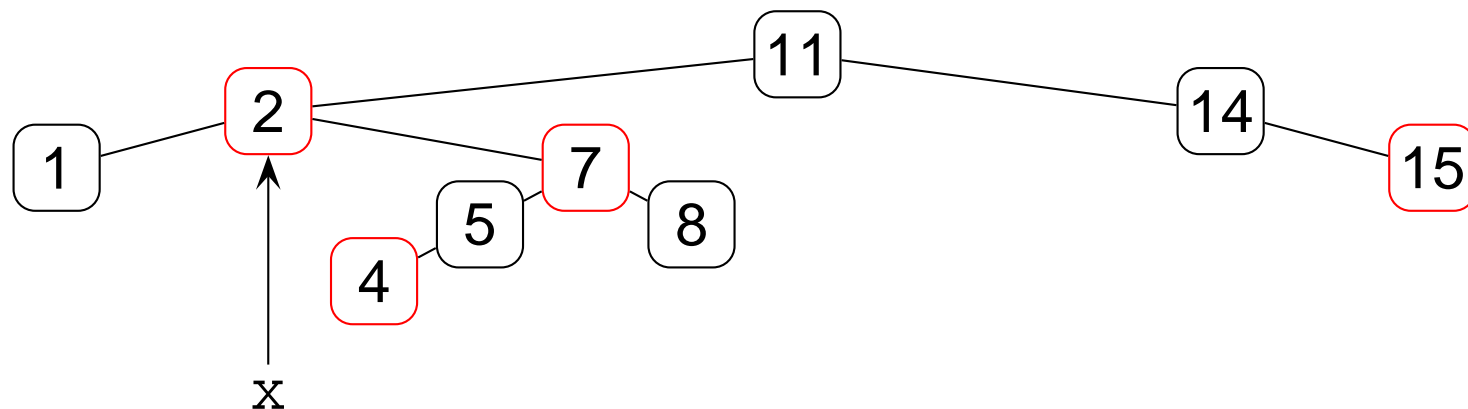
`p(y) ← p(x)`

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`

`left(y) ← x, p(x) ← y`



Red-Black Insert (8) | Red-Black Insert Fixup (6) | Left Rotate (1)

lRot(Tree T, Node x):

$y \leftarrow \text{right}(x)$

$\text{right}(x) \leftarrow \text{left}(y)$

if $\text{left}(y) \neq \text{nil}(T)$ **then** $p(\text{left}(y)) \leftarrow x$

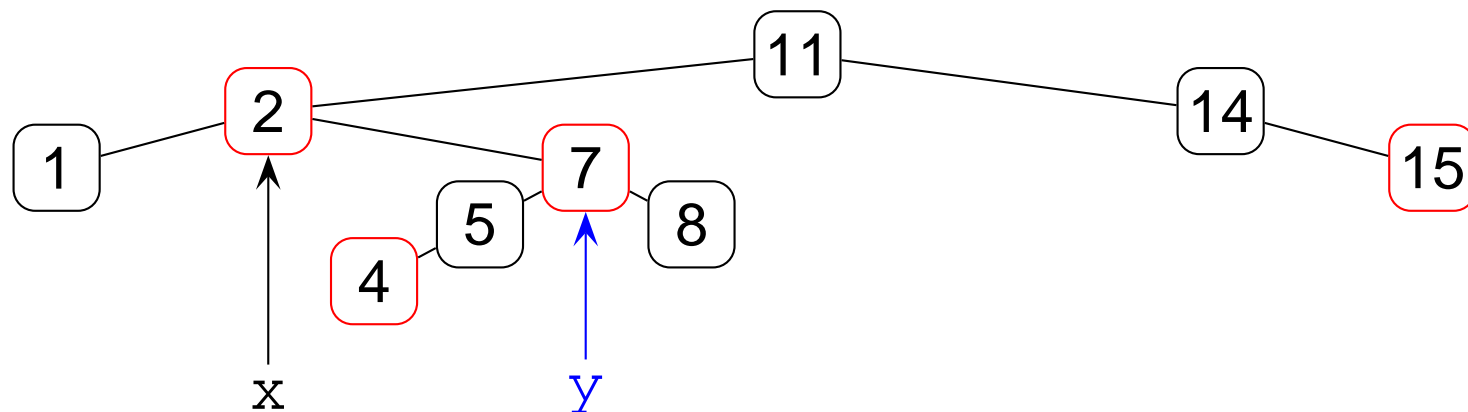
$p(y) \leftarrow p(x)$

if $p(x) = \text{nil}(T)$ **then** $\text{root}(T) \leftarrow y$

else if $x = \text{left}(p(x))$ **then** $\text{left}(p(x)) \leftarrow y$

else $\text{right}(p(x)) \leftarrow y$

$\text{left}(y) \leftarrow x, p(x) \leftarrow y$



Red-Black Insert (8) | Red-Black Insert Fixup (6) | Left Rotate (2)

lRot(Tree T, Node x):

$y \leftarrow \text{right}(x)$

$\text{right}(x) \leftarrow \text{left}(y)$

if $\text{left}(y) \neq \text{nil}(T)$ **then** $p(\text{left}(y)) \leftarrow x$

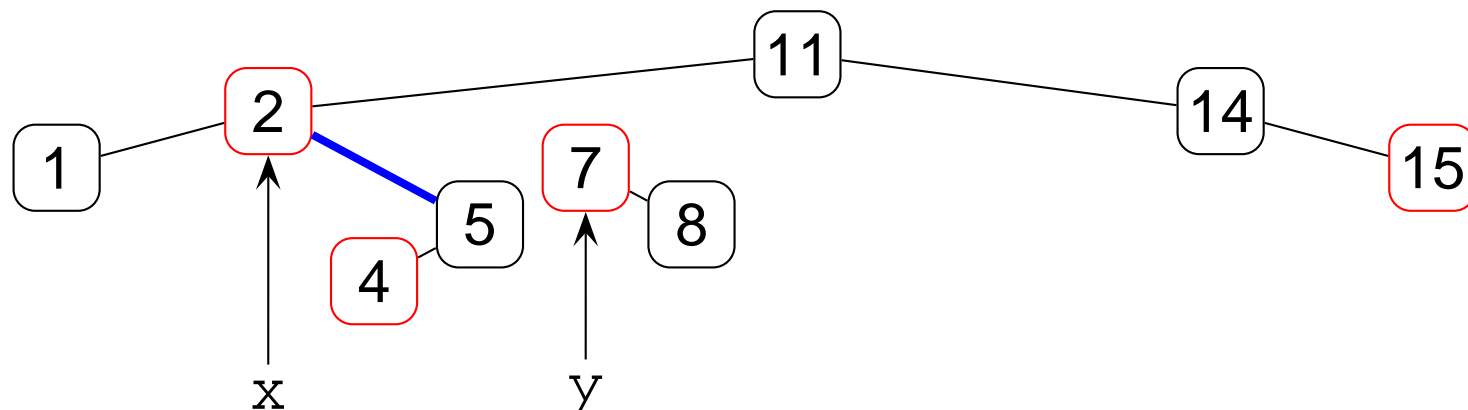
$p(y) \leftarrow p(x)$

if $p(x) = \text{nil}(T)$ **then** $\text{root}(T) \leftarrow y$

else if $x = \text{left}(p(x))$ **then** $\text{left}(p(x)) \leftarrow y$

else $\text{right}(p(x)) \leftarrow y$

$\text{left}(y) \leftarrow x, p(x) \leftarrow y$



Red-Black Insert (8) | Red-Black Insert Fixup (6) | Left Rotate (done)

lRot(Tree T, Node x):

$y \leftarrow \text{right}(x)$

$\text{right}(x) \leftarrow \text{left}(y)$

if $\text{left}(y) \neq \text{nil}(T)$ **then** $p(\text{left}(y)) \leftarrow x$

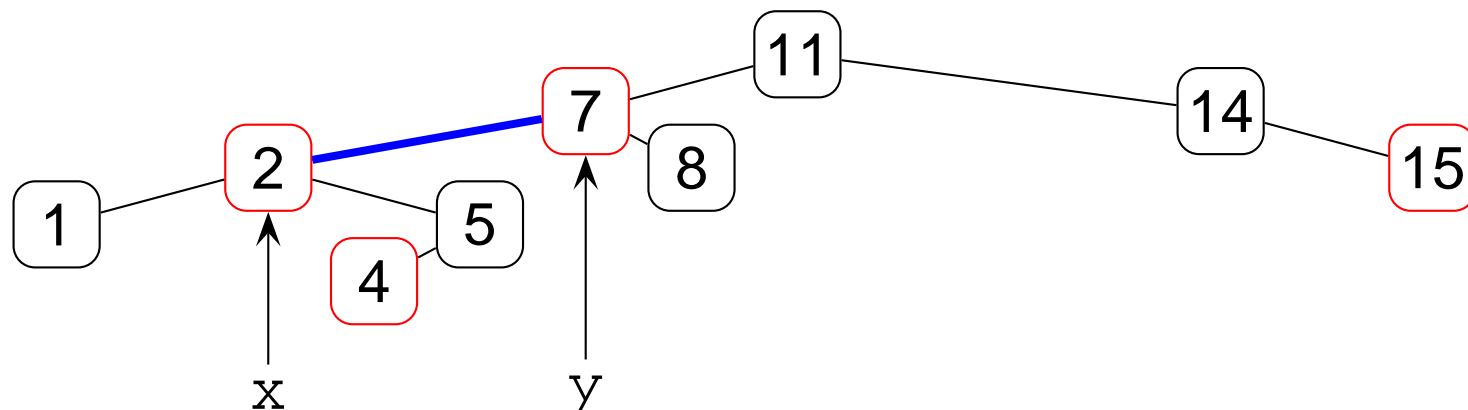
$p(y) \leftarrow p(x)$

if $p(x) = \text{nil}(T)$ **then** $\text{root}(T) \leftarrow y$

else if $x = \text{left}(p(x))$ **then** $\text{left}(p(x)) \leftarrow y$

else $\text{right}(p(x)) \leftarrow y$

$\text{left}(y) \leftarrow x, p(x) \leftarrow y$



Red-Black Insert (8) | Red-Black Insert Fixup (6)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

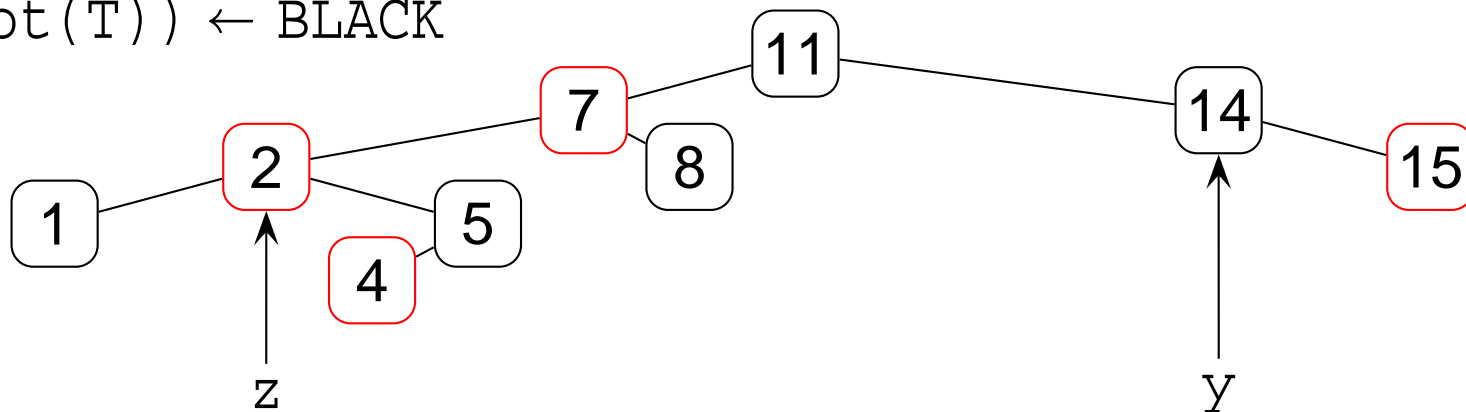
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (8)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

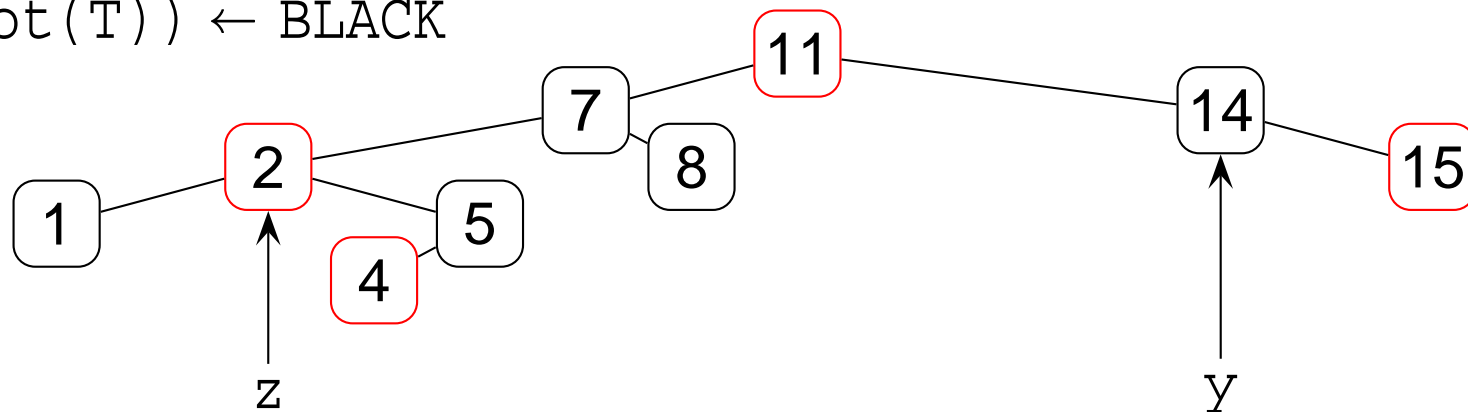
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (8) | Right Rotate

`rRot(Tree T, Node x):`

`y ← left(x)`

`left(x) ← right(y)`

if `right(y) ≠ nil(T)` **then** `p(right(y)) ← x`

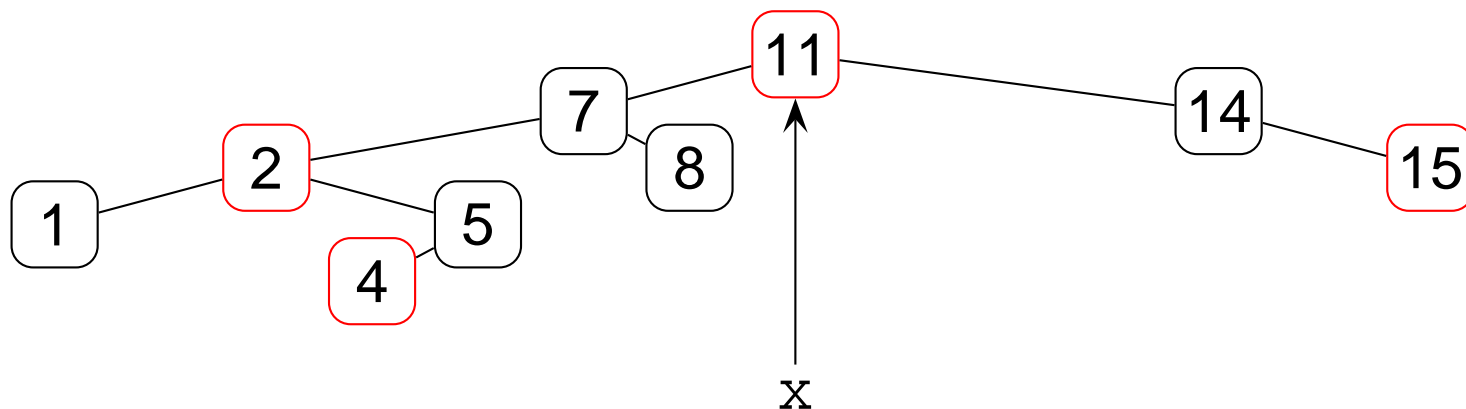
`p(y) ← p(x)`

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`

`right(y) ← x, p(x) ← y`



Red-Black Insert (8) | Red-Black Insert Fixup (8) | Right Rotate (1)

rRot(Tree T, Node x):

$y \leftarrow \text{left}(x)$

$\text{left}(x) \leftarrow \text{right}(y)$

if $\text{right}(y) \neq \text{nil}(T)$ **then** $p(\text{right}(y)) \leftarrow x$

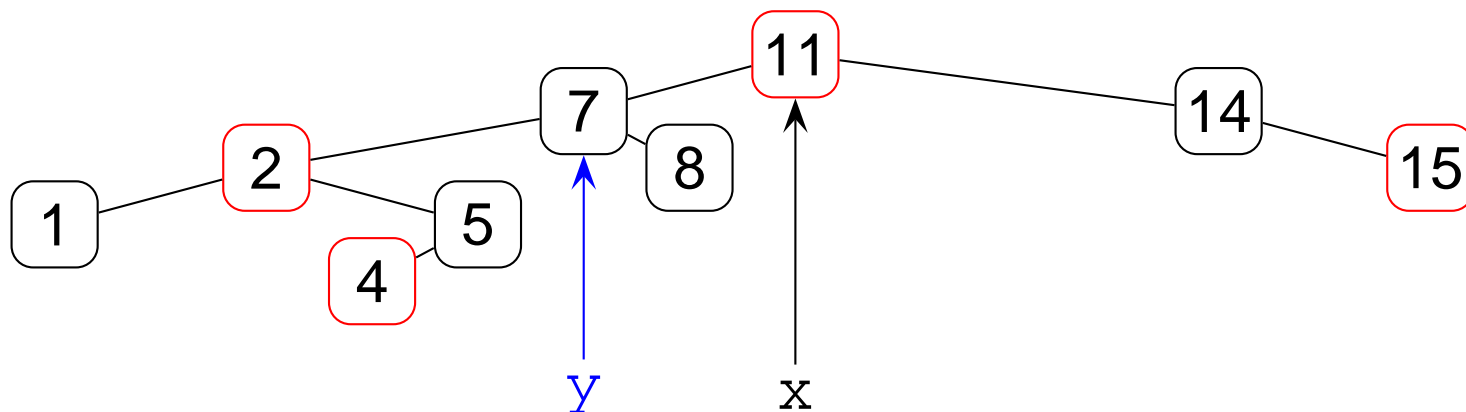
$p(y) \leftarrow p(x)$

if $p(x) = \text{nil}(T)$ **then** $\text{root}(T) \leftarrow y$

else if $x = \text{left}(p(x))$ **then** $\text{left}(p(x)) \leftarrow y$

else $\text{right}(p(x)) \leftarrow y$

$\text{right}(y) \leftarrow x, p(x) \leftarrow y$



Red-Black Insert (8) | Red-Black Insert Fixup (8) | Right Rotate (2)

rRot(Tree T, Node x):

$y \leftarrow \text{left}(x)$

$\text{left}(x) \leftarrow \text{right}(y)$

if $\text{right}(y) \neq \text{nil}(T)$ **then** $p(\text{right}(y)) \leftarrow x$

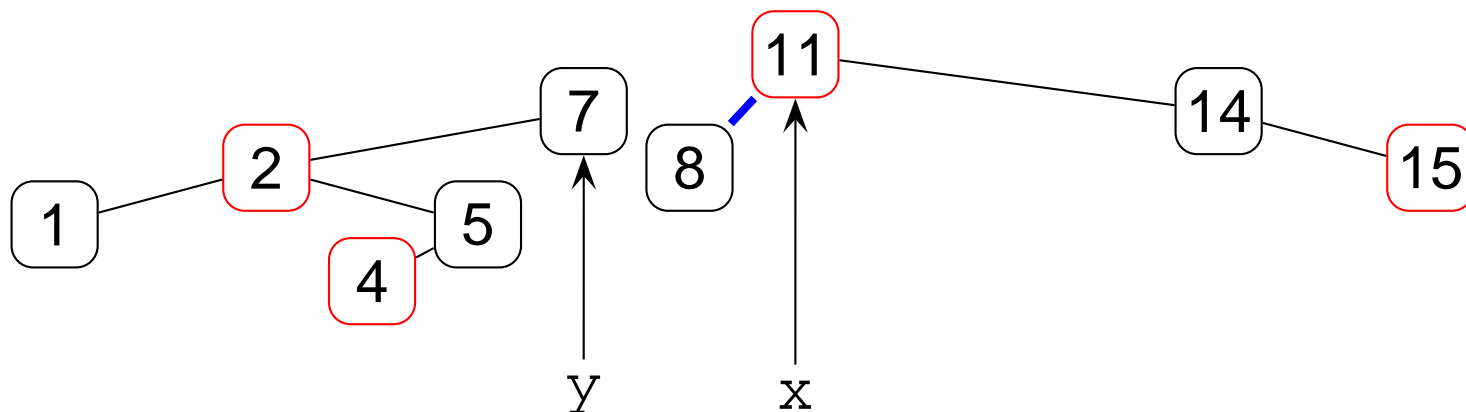
$p(y) \leftarrow p(x)$

if $p(x) = \text{nil}(T)$ **then** $\text{root}(T) \leftarrow y$

else if $x = \text{left}(p(x))$ **then** $\text{left}(p(x)) \leftarrow y$

else $\text{right}(p(x)) \leftarrow y$

$\text{right}(y) \leftarrow x, p(x) \leftarrow y$



Red-Black Insert (8) | Red-Black Insert Fixup (8) | Right Rotate (3)

rRot(Tree T, Node x):

$y \leftarrow \text{left}(x)$

$\text{left}(x) \leftarrow \text{right}(y)$

if $\text{right}(y) \neq \text{nil}(T)$ **then** $p(\text{right}(y)) \leftarrow x$

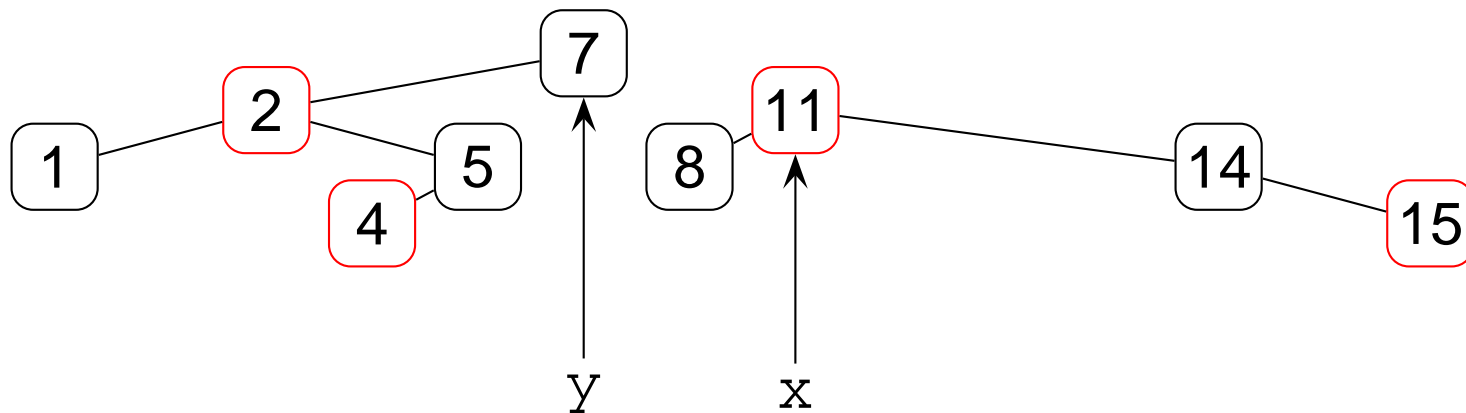
$p(y) \leftarrow p(x)$

if $p(x) = \text{nil}(T)$ **then** $\text{root}(T) \leftarrow y$

else if $x = \text{left}(p(x))$ **then** $\text{left}(p(x)) \leftarrow y$

else $\text{right}(p(x)) \leftarrow y$

$\text{right}(y) \leftarrow x, p(x) \leftarrow y$



Red-Black Insert (8) | Red-Black Insert Fixup (8) | Right Rotate (done)

rRot(Tree T, Node x):

$y \leftarrow \text{left}(x)$

$\text{left}(x) \leftarrow \text{right}(y)$

if $\text{right}(y) \neq \text{nil}(T)$ **then** $p(\text{right}(y)) \leftarrow x$

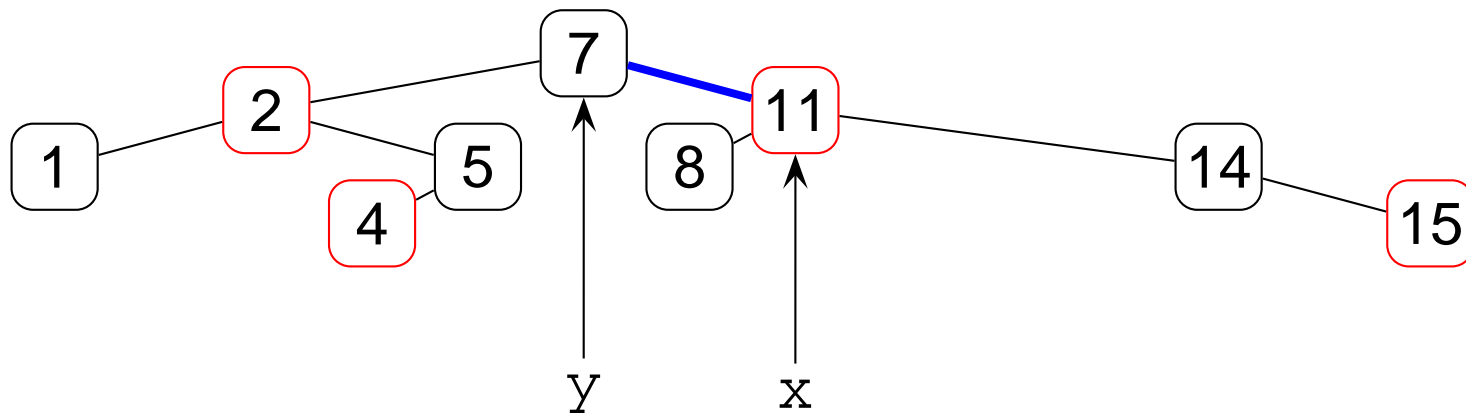
$p(y) \leftarrow p(x)$

if $p(x) = \text{nil}(T)$ **then** $\text{root}(T) \leftarrow y$

else if $x = \text{left}(p(x))$ **then** $\text{left}(p(x)) \leftarrow y$

else $\text{right}(p(x)) \leftarrow y$

$\text{right}(y) \leftarrow x, p(x) \leftarrow y$



Red-Black Insert (8) | Red-Black Insert Fixup (8)

rbInsertFix(Tree T, Node z):

while col(p(z))=RED

if p(z)=left(p(p(z))) **then**

 y ← right(p(p(z)))

if col(y)=RED **then**

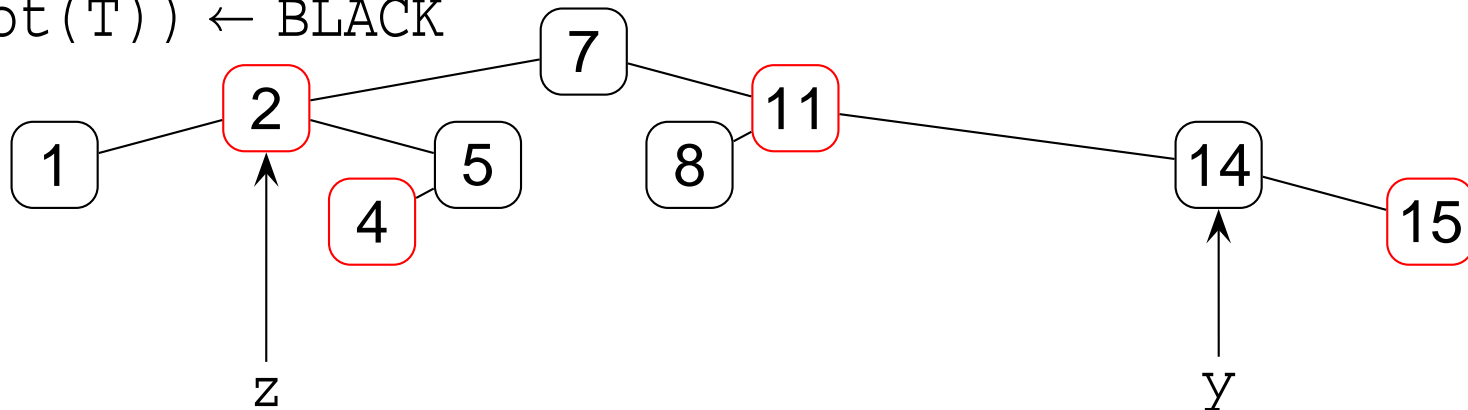
 col(p(z)) ← col(y) ← BLK, col(p(p(z))) ← RED, z ← p(p(z))

else if z=right(p(z)) **then** z ← p(z), lRot(T,z)

 col(p(z)) ← BLK, col(p(p(z))) ← RED, rRot(T,p(p(z)))

else ... (same as **then** clause, with 'right' and 'left' exchanged)

col(root(T)) ← BLACK



Red-Black Insert (8) | Red-Black Insert Fixup (done)

rbInsertFix(Tree T, Node z):

while $\text{col}(p(z)) = \text{RED}$

if $p(z) = \text{left}(p(p(z)))$ **then**

$y \leftarrow \text{right}(p(p(z)))$

if $\text{col}(y) = \text{RED}$ **then**

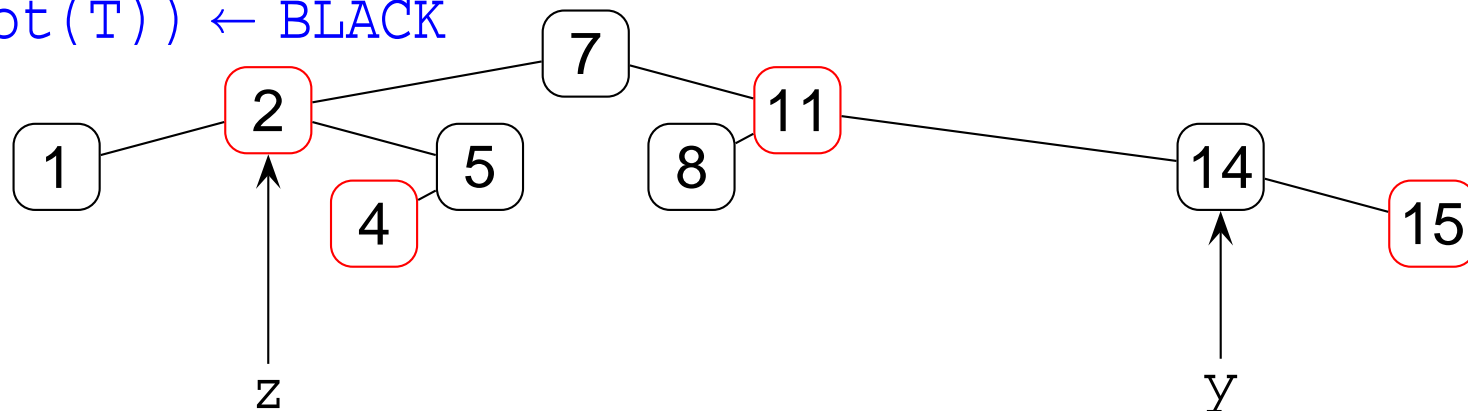
$\text{col}(p(z)) \leftarrow \text{col}(y) \leftarrow \text{BLK}, \text{col}(p(p(z))) \leftarrow \text{RED}, z \leftarrow p(p(z))$

else if $z = \text{right}(p(z))$ **then** $z \leftarrow p(z), \text{lRot}(T, z)$

$\text{col}(p(z)) \leftarrow \text{BLK}, \text{col}(p(p(z))) \leftarrow \text{RED}, \text{rRot}(T, p(p(z)))$

else ... (same as **then** clause, with 'right' and 'left' exchanged)

$\text{col}(\text{root}(T)) \leftarrow \text{BLACK}$



Red-Black Insert (done)

rbInsert(Tree T, Node z):

```
loop from y ← nil(T), x ← root(T) while x ≠ nil(T)  
  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
```

```
  else if key(z) < key(y) then left(y) ← z  
  else right(y) ← z
```

```
left(z) ← right(z) ← nil(T), col(z) ← RED, rbInsertFix(T, z)
```

