CH-231-A

Algorithms and Data Structures

ADS

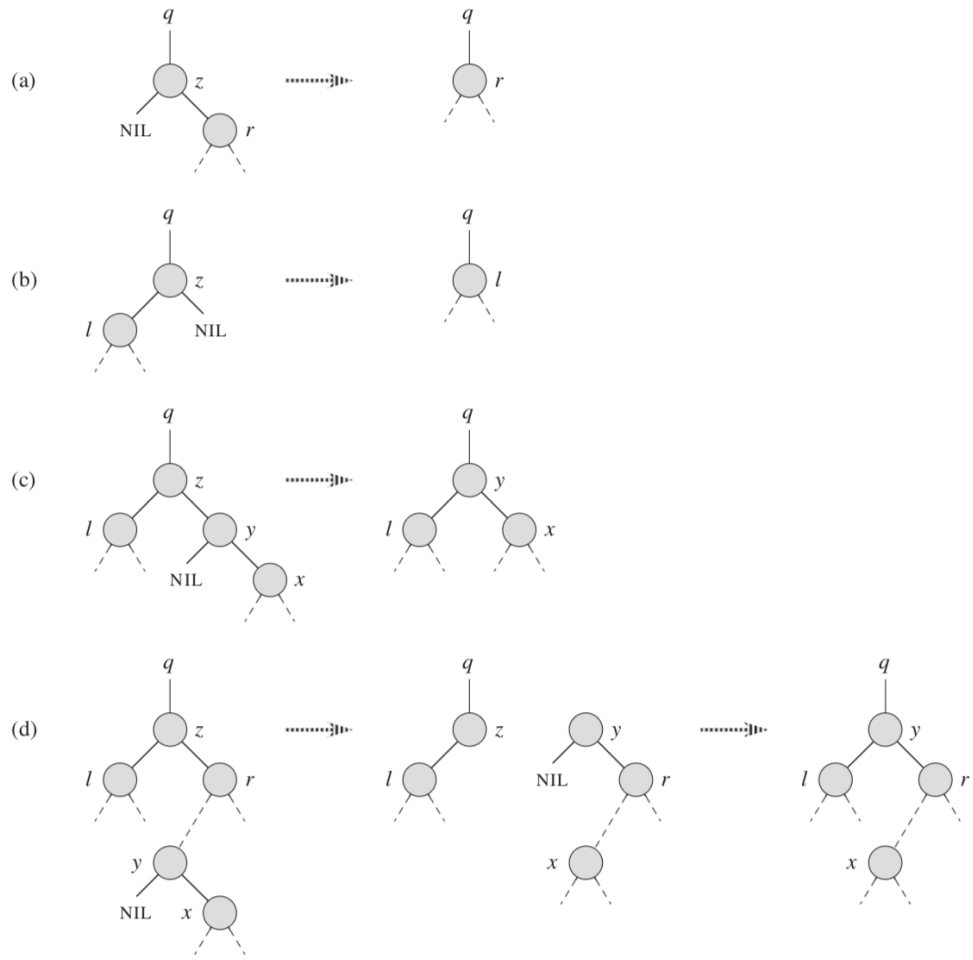
Lecture 26

Dr. Kinga Lipskoch

Spring 2020

Deletion (Remember BST)

```
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
```



Deletion (RB) (1)

```
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)
```

Deletion (RB) (2)

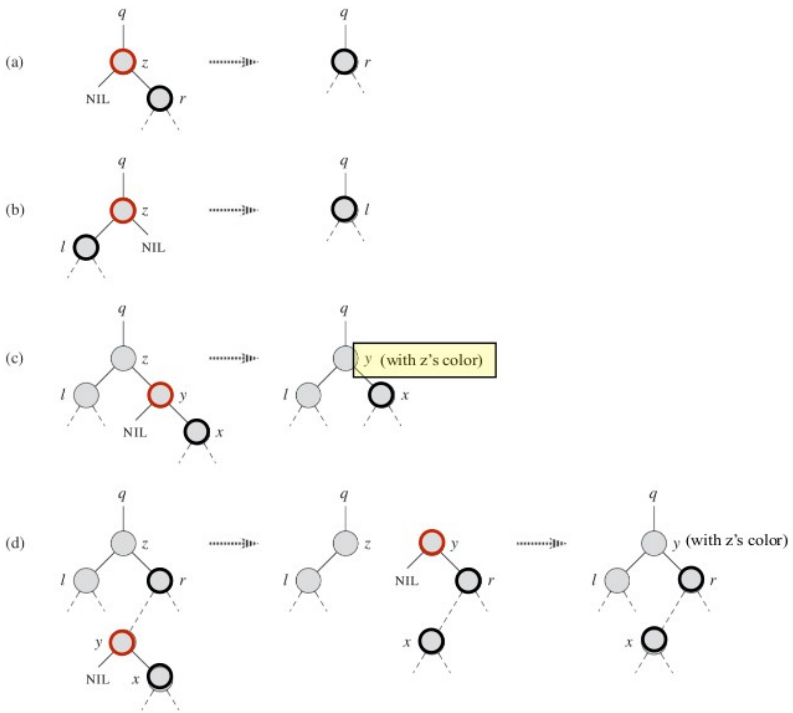
- node y
  - either removed (a/b)
  - or moved in the tree (c/d)
  - y-original-color
- node x
  - the node that moves into y's original position
  - x.p points to y's original parent (since it moves into y's position, note special case in 12/13)

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)
```

Deletion (RB) (3)

- y-original-color == red

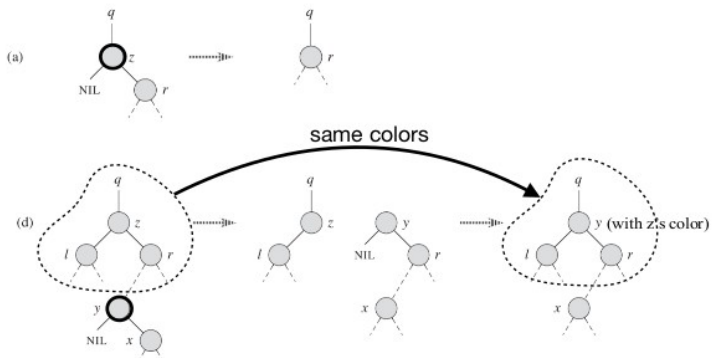


RB-DELETE(T, z)

```
 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)
```

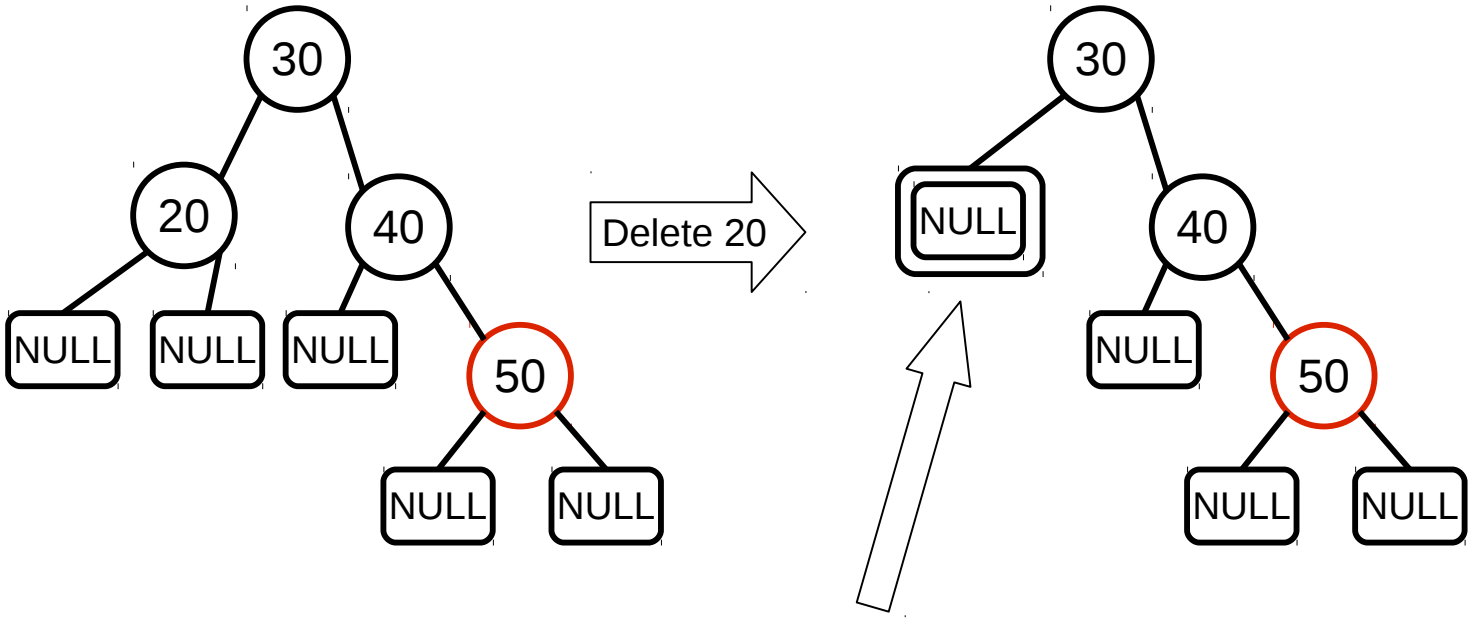
Deletion (RB) (4)

- y-original-color == red
  - no problem
- y-original-color == black
  - violations might occur (2,4,5)
  - main idea to fix
    - x gets an "**extra black**" & needs to get rid of it
  - 4 cases




```
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)
```

Extra Black or Double Black Node

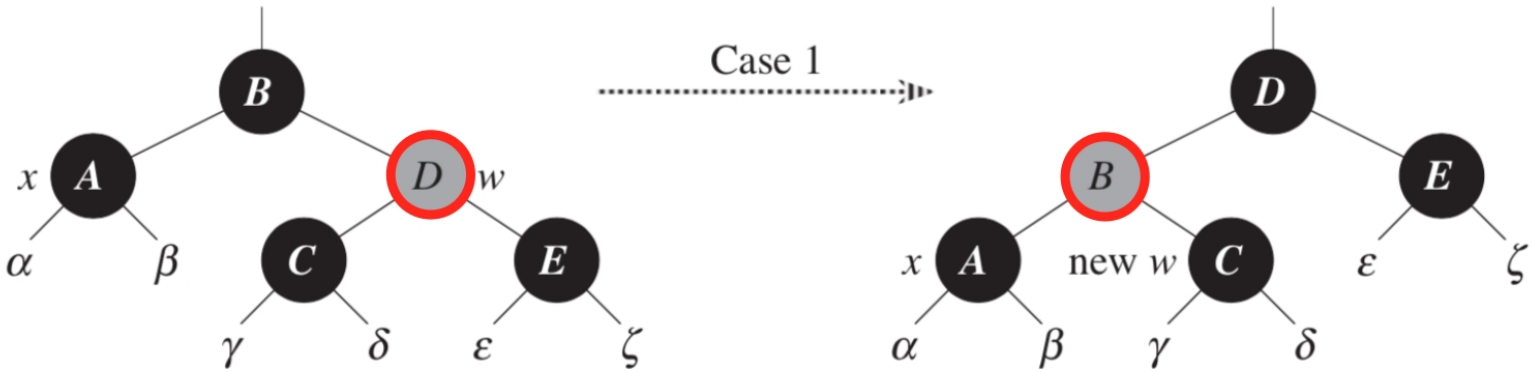


Child caries „extra black“ information
also called „double black“ node

Fixing Red-Black Tree Properties (1)

Case 1: x's sibling w is red.

Transform to Case 2, 3, or 4 by left rotation and changing colors of nodes B and D.



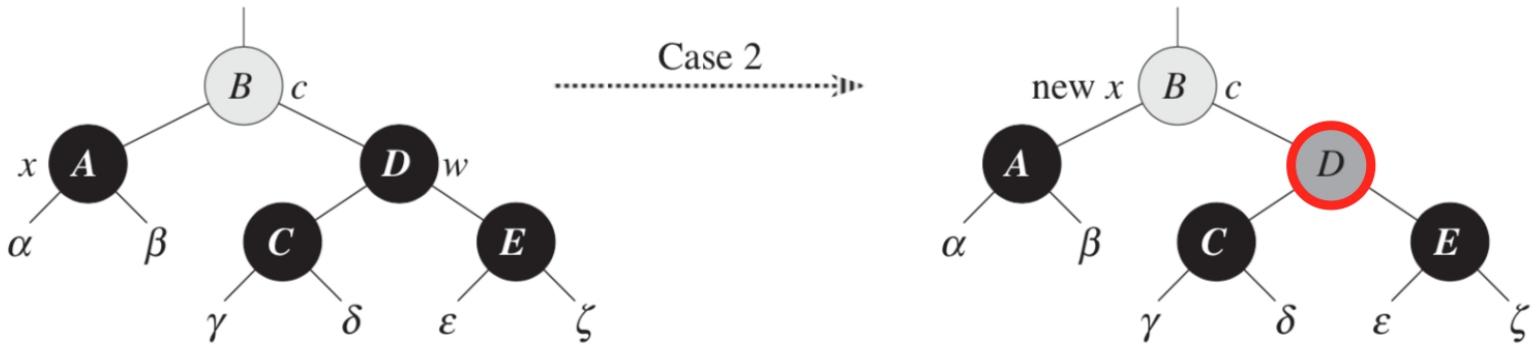
x = node with extra black

w = x's sibling

if $w.color$ == RED
  $w.color = \text{BLACK}$
  $x.p.color = \text{RED}$
  LEFT-ROTATE$(T, x.p)$
  $w = x.p.right$

Fixing Red-Black Tree Properties (2)

Case 2: x's sibling w is black and the children of w are black. Set color of w to red and propagate upwards.



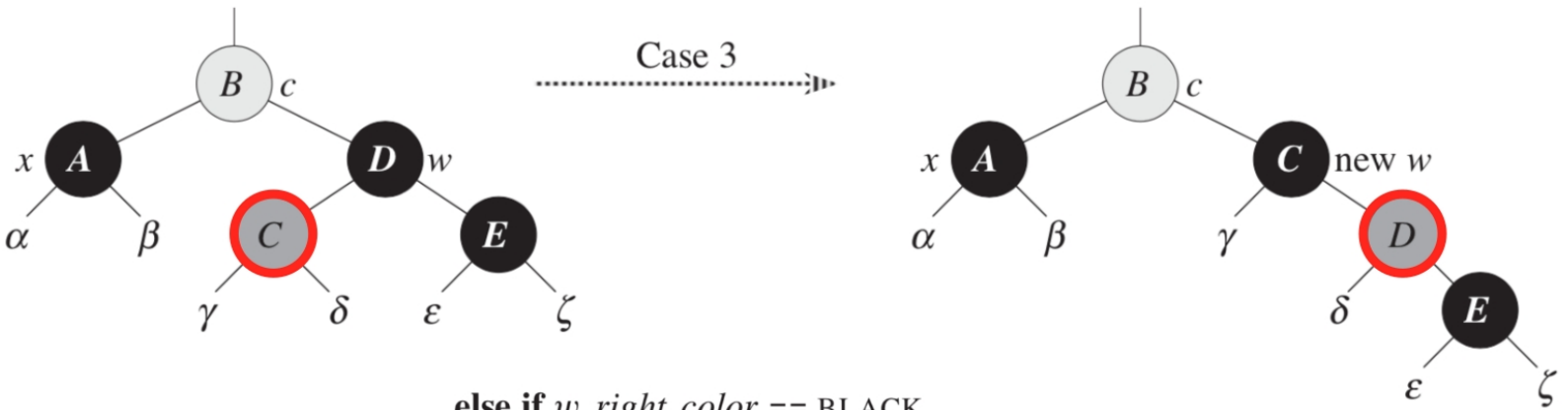
x = node with extra black
w = x's sibling
c = color of the node

if $w.left.color$ == BLACK and $w.right.color$ == BLACK
$w.color = \text{RED}$
$x = x.p$

Fixing Red-Black Tree Properties (3)

Case 3: x's sibling w is black and the left child of w is red, while the right child of w is black.

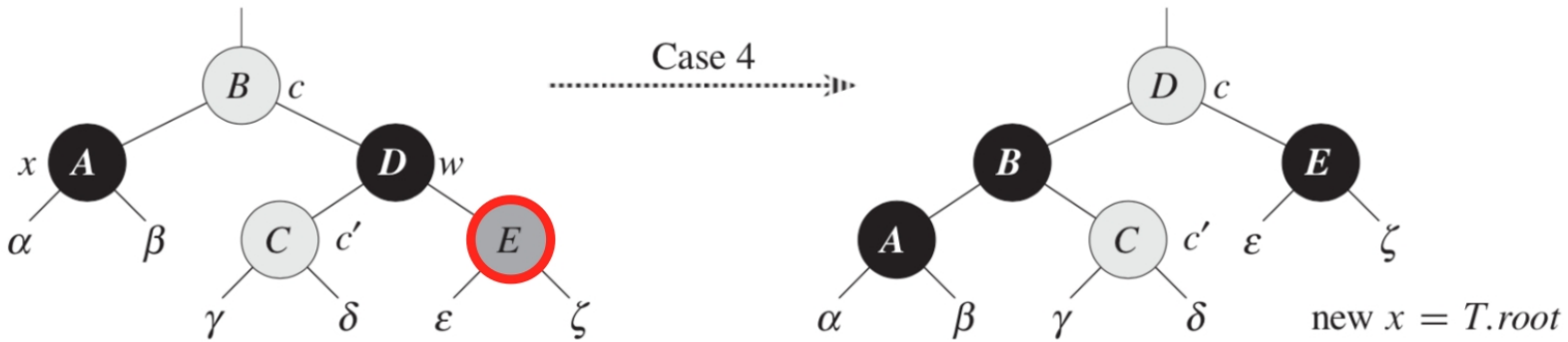
Transform to Case 4 by right rotation and changing colors of nodes C and D.



else if $w.right.color$ == BLACK
$w.left.color$ = BLACK
$w.color$ = RED
RIGHT-ROTATE(T, w)
$w = x.p.right$

Fixing Red-Black Tree Properties (4)

Case 4: x's sibling w is black and the right child of w is red. Perform a left-rotate and change colors of B, D, and E. Then, the loop terminates.



$w.color = x.p.color$
$x.p.color = \text{BLACK}$
$w.right.color = \text{BLACK}$
$\text{LEFT-ROTATE}(T, x.p)$

Fixing Red-Black Tree Properties (5)

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

Time complexity: $O(h) = O(\lg n)$

Conclusion

Modifying operations on red-black trees can be executed in $O(\lg n)$ time.