Data structure [A13]

김종규, PhD

2017-05-29

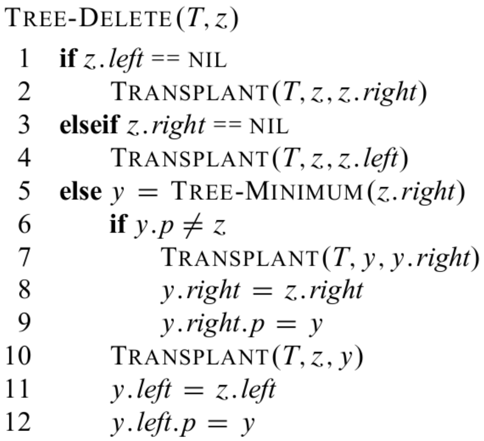
Reviews

- red-black tree insert
  - case 3: black

Outline

- red-black tree delete

Tree-Delete



```
TREE-DELETE($T, z$)
if $z.left$ == NIL
    TRANSPLANT($T, z, z.right$)
elseif $z.right$ == NIL
    TRANSPLANT($T, z, z.left$)
else $y$ = TREE-MINIMUM($z.right$)
    if $y.p \neq z$
        TRANSPLANT($T, y, y.right$)
        $y.right = z.right$
        $y.right.p = y$
    TRANSPLANT($T, z, y$)
    $y.left = z.left$
    $y.left.p = y$
```

그림: Tree-Delete

RB-Tree: Delete a node

- What is deleting a node x, y, z

RB-Delete

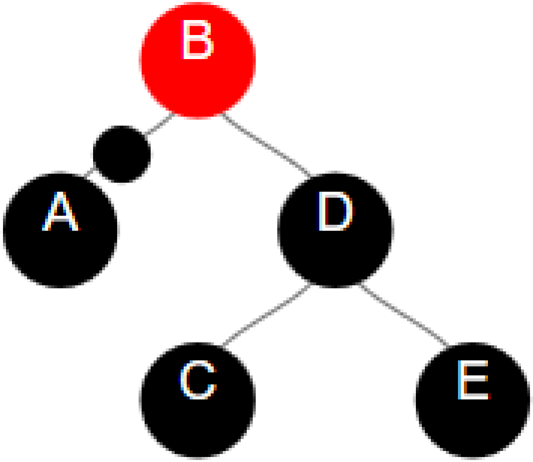


그림: RB-Delete (a)

RB-Delete

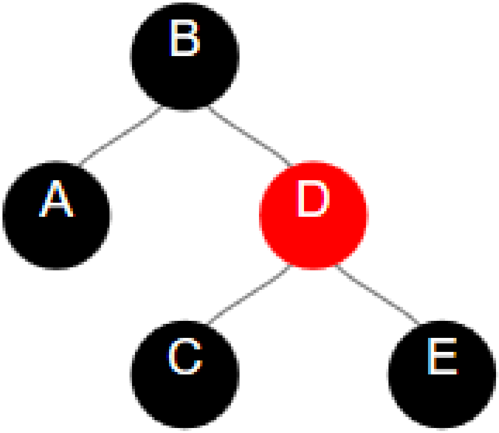


그림: RB-Delete (b)

RB-Delete

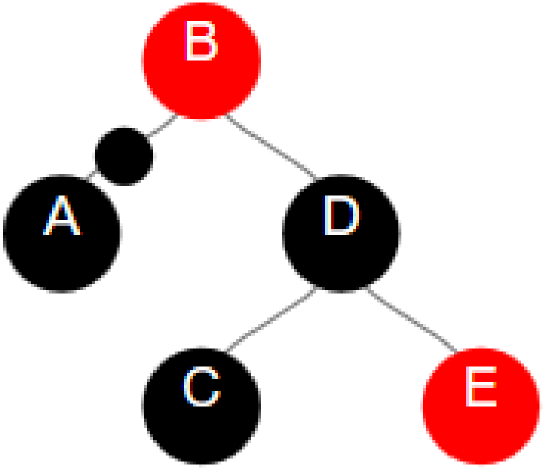


그림: RB-Delete (e)

RB-Delete

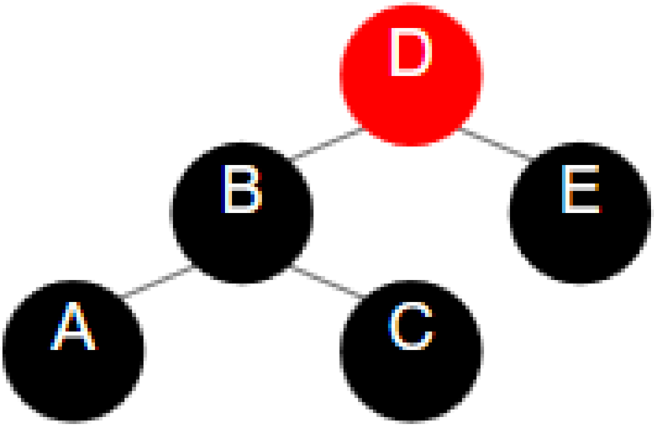


그림: RB-Delete (f)

RB-Delete

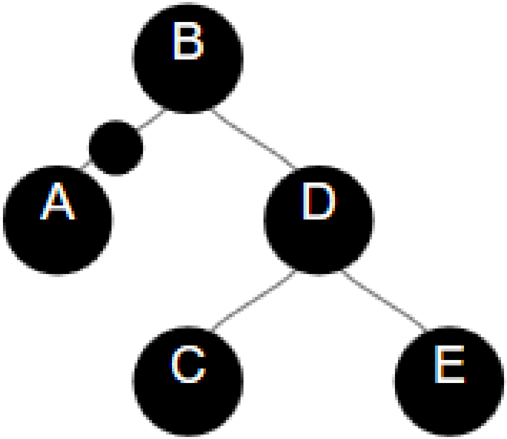


그림: RB-Delete (c)

RB-Delete

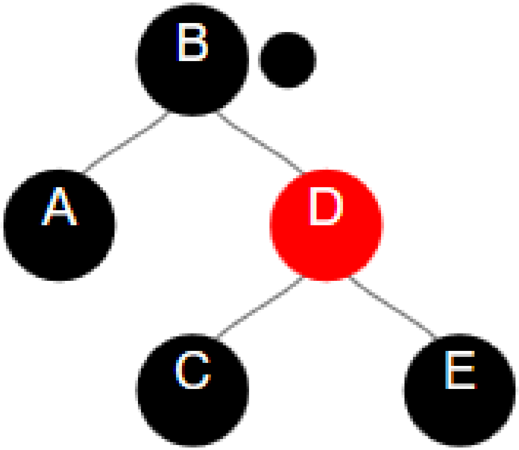


그림: RB-Delete (d)

RB-Tree: Delete a node

- Deleting a red node

RB-Tree: Delete a node

- Deleting a black node

Case 4: termination condition

- *x*'s sibling *w* is black and right child is red

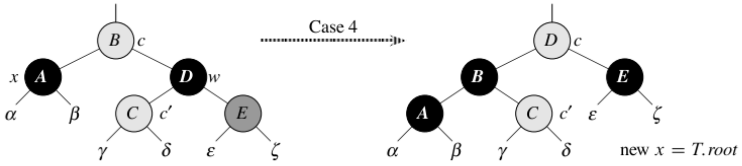


그림: RB-Fixup: Case 4

Case 3: Transform to Case 4

- *x*'s sibling *w* is black and left child is red and right child is black

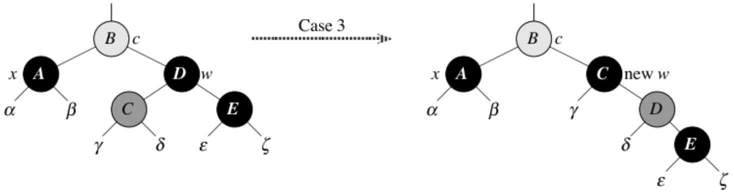


그림: RB-Fixup: Case 3

Case 2: Move to parent

- x's sibling w is black and both children are black

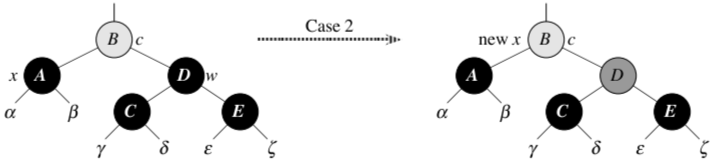


그림: RB-Fixup: Case 2

Case 1: Transform to Case 2

- *x*'s sibling *w* is red

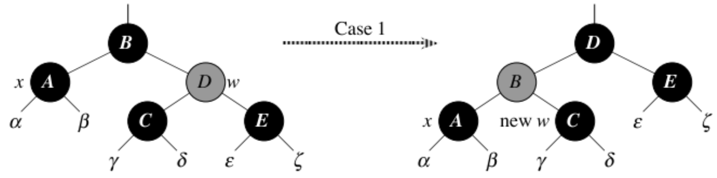
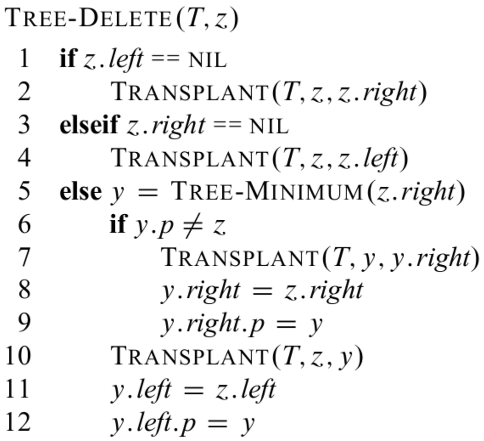


그림: RB-Fixup: Case 1

Homework

- Implement red-black tree
- Submit using `github`

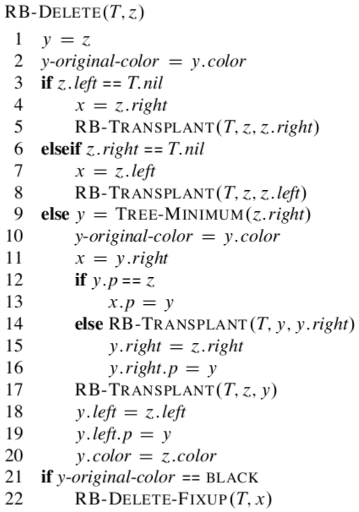
Tree-Delete



```
TREE-DELETE($T, z$)
if $z.left$ == NIL
    TRANSPLANT($T, z, z.right$)
elseif $z.right$ == NIL
    TRANSPLANT($T, z, z.left$)
else $y = $ TREE-MINIMUM($z.right$)
    if $y.p \neq z$
        TRANSPLANT($T, y, y.right$)
        $y.right = z.right$
        $y.right.p = y$
    TRANSPLANT($T, z, y$)
    $y.left = z.left$
    $y.left.p = y$
```

그림: Tree-Delete

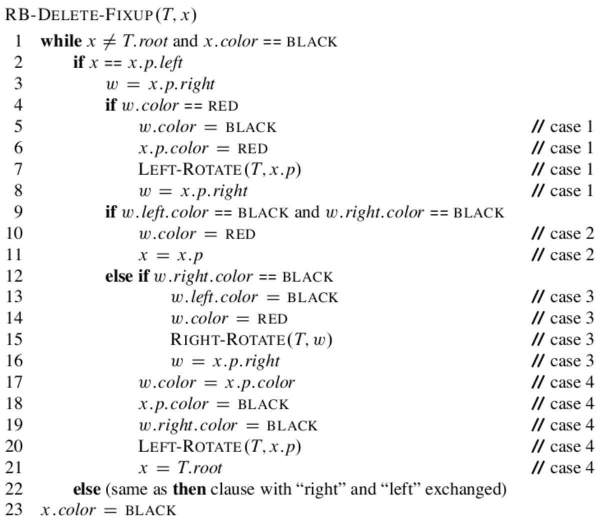
RB-Delete



```
RB-DELETE($T, z$)
$y = z$
$y$-original-color $= y.color$
if $z.left == T.nil$
    $x = z.right$
    RB-TRANSPLANT($T, z, z.right$)
elseif $z.right == T.nil$
    $x = z.left$
    RB-TRANSPLANT($T, z, z.left$)
else $y =$ TREE-MINIMUM($z.right$)
    $y$-original-color $= y.color$
    $x = y.right$
    if $y.p == z$
        $x.p = y$
    else RB-TRANSPLANT($T, y, y.right$)
        $y.right = z.right$
        $y.right.p = y$
    RB-TRANSPLANT($T, z, y$)
    $y.left = z.left$
    $y.left.p = y$
    $y.color = z.color$
if $y$-original-color == BLACK
    RB-DELETE-FIXUP($T, x$)
```

그림: RB-delete

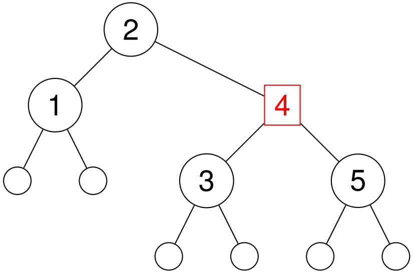
RB-Fixup



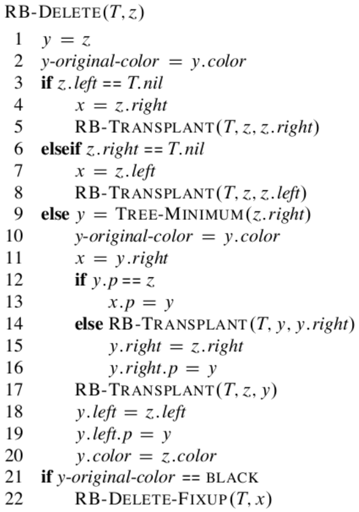
```
RB-DELETE-FIXUP(T, x)
while $x \neq T.root$ and $x.color$ == BLACK
    if $x == x.p.left$
        $w = x.p.right$
        if $w.color$ == RED
            $w.color$ = BLACK                                  // case 1
            $x.p.color$ = RED                                  // case 1
            LEFT-ROTATE(T, x.p)                                // case 1
            $w = x.p.right$                                    // case 1
        if $w.left.color$ == BLACK and $w.right.color$ == BLACK
            $w.color$ = RED                                    // case 2
            $x = x.p$                                          // case 2
        else if $w.right.color$ == BLACK
                $w.left.color$ = BLACK                         // case 3
                $w.color$ = RED                                // case 3
                RIGHT-ROTATE(T, w)                             // case 3
                $w = x.p.right$                                // case 3
            $w.color = x.p.color$                              // case 4
            $x.p.color$ = BLACK                                // case 4
            $w.right.color$ = BLACK                            // case 4
            LEFT-ROTATE(T, x.p)                                // case 4
            $x = T.root$                                       // case 4
    else (same as then clause with "right" and "left" exchanged)
$x.color$ = BLACK
```

그림: RB-Fixup

Example: red-black tree



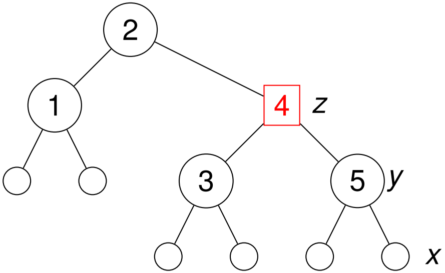
RB-Delete

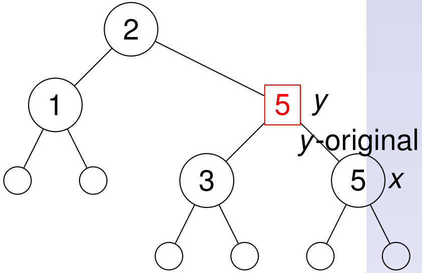


```
RB-DELETE($T, z$)
  $y = z$
  $y$-original-color $= y.color$
  if $z.left == T.nil$
      $x = z.right$
      RB-TRANSPLANT($T, z, z.right$)
  elseif $z.right == T.nil$
      $x = z.left$
      RB-TRANSPLANT($T, z, z.left$)
  else $y$ = TREE-MINIMUM($z.right$)
      $y$-original-color $= y.color$
      $x = y.right$
      if $y.p == z$
          $x.p = y$
      else RB-TRANSPLANT($T, y, y.right$)
          $y.right = z.right$
          $y.right.p = y$
      RB-TRANSPLANT($T, z, y$)
      $y.left = z.left$
      $y.left.p = y$
      $y.color = z.color$
  if $y$-original-color == BLACK
      RB-DELETE-FIXUP($T, x$)
```

그림: RB-delete

Example: Delete 4





Case 2: Move to parent

- *x*'s sibling *w* is black and both children are black

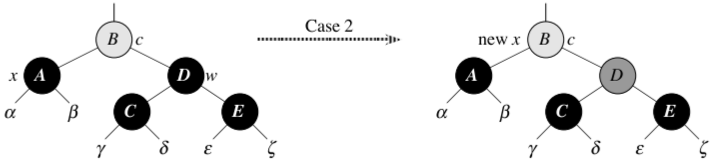
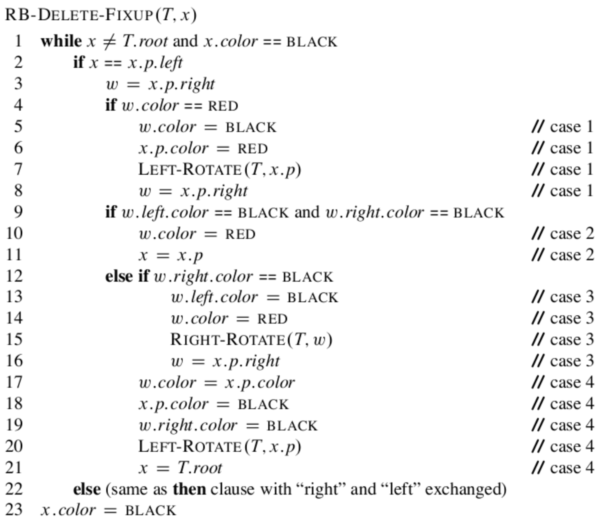


그림: RB-Fixup: Case 2

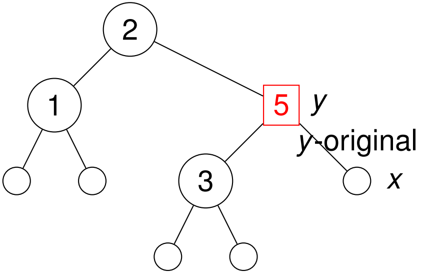
RB-Fixup

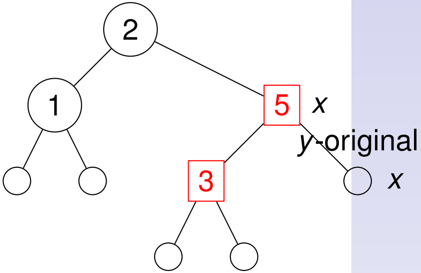


```
RB-DELETE-FIXUP$(T, x)$
while $x \neq T.root$ and $x.color == \text{BLACK}$
    if $x == x.p.left$
        $w = x.p.right$
        if $w.color == \text{RED}$
            $w.color = \text{BLACK}$                              // case 1
            $x.p.color = \text{RED}$                              // case 1
            LEFT-ROTATE$(T, x.p)$                                 // case 1
            $w = x.p.right$                                       // case 1
        if $w.left.color == \text{BLACK}$ and $w.right.color == \text{BLACK}$
            $w.color = \text{RED}$                                // case 2
            $x = x.p$                                             // case 2
        else if $w.right.color == \text{BLACK}$
                $w.left.color = \text{BLACK}$                     // case 3
                $w.color = \text{RED}$                            // case 3
                RIGHT-ROTATE$(T, w)$                              // case 3
                $w = x.p.right$                                   // case 3
            $w.color = x.p.color$                                 // case 4
            $x.p.color = \text{BLACK}$                            // case 4
            $w.right.color = \text{BLACK}$                        // case 4
            LEFT-ROTATE$(T, x.p)$                                 // case 4
            $x = T.root$                                          // case 4
    else (same as then clause with "right" and "left" exchanged)
$x.color = \text{BLACK}$
```

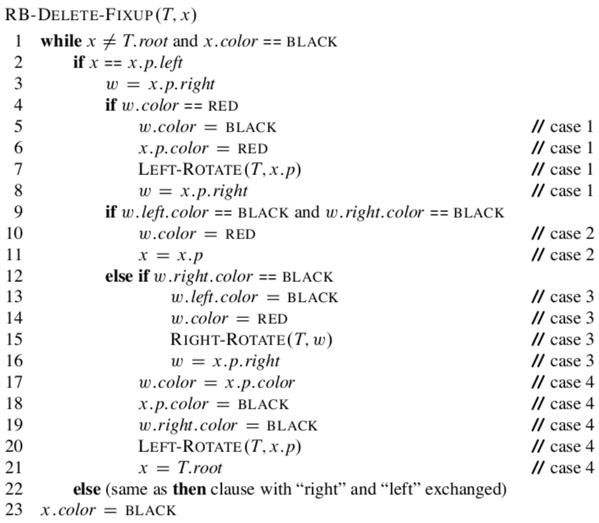
그림: RB-Fixup

Example: Delete 4, fix-up





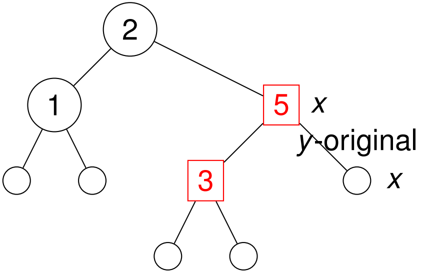
RB-Fixup

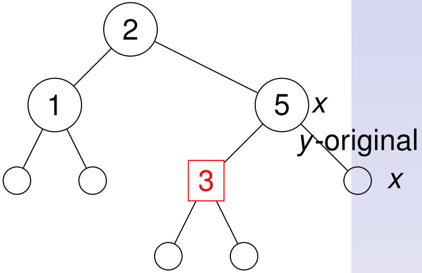


```
RB-DELETE-FIXUP$(T, x)$
while $x \neq T.root$ and $x.color$ == BLACK
    if $x == x.p.left$
        $w = x.p.right$
        if $w.color$ == RED
            $w.color$ = BLACK                                  // case 1
            $x.p.color$ = RED                                  // case 1
            LEFT-ROTATE$(T, x.p)$                              // case 1
            $w = x.p.right$                                    // case 1
        if $w.left.color$ == BLACK and $w.right.color$ == BLACK
            $w.color$ = RED                                    // case 2
            $x = x.p$                                          // case 2
        else if $w.right.color$ == BLACK
                $w.left.color$ = BLACK                         // case 3
                $w.color$ = RED                                // case 3
                RIGHT-ROTATE$(T, w)$                           // case 3
                $w = x.p.right$                                // case 3
            $w.color = x.p.color$                              // case 4
            $x.p.color$ = BLACK                                // case 4
            $w.right.color$ = BLACK                            // case 4
            LEFT-ROTATE$(T, x.p)$                              // case 4
            $x = T.root$                                       // case 4
    else (same as then clause with "right" and "left" exchanged)
$x.color$ = BLACK
```

그림: RB-Fixup

Example: Delete 4, fix-up





Wrap-up

- We reviewed delete of red-black tree
- This is an ordinary tree deletion for *z*
  - If *z* does not have both children, removed node *y* becomes *z*
  - To delete *z* which has both children, we transplant *y* to *z*
  - *x* is the right child of *y* and transplanted to *y*
- When the removed node *y* is black, we fix-up the tree to satisfy red-black properties