Red-Black Trees

Chapters 13

Binary Search Trees Review

Balanced Trees

- Why do we want to balance trees?
- Red-Black Trees are an example of balanced trees
- Other balanced trees:
  - AVL trees
  - B-trees
  - 2-3 trees

Red-Black Tree

- BST data structure with extra color field for each node, satisfying the red-black properties:
  1. Every node is either red or black.
  2. The root is black.
  3. Every leaf is black.
  4. If a node is red, both children are black.
  5. Every path from node to descendent leaf contain the same number of black nodes.

Example

- Attributes of nodes:
  - key
  - left
  - right
  - p (parent)
  - color
- Note the use of the sentinel T.nil
  - Parent of the root is T.nil
  - All leaves are T.nil – no data in the leaves!

Properties of RB-Trees

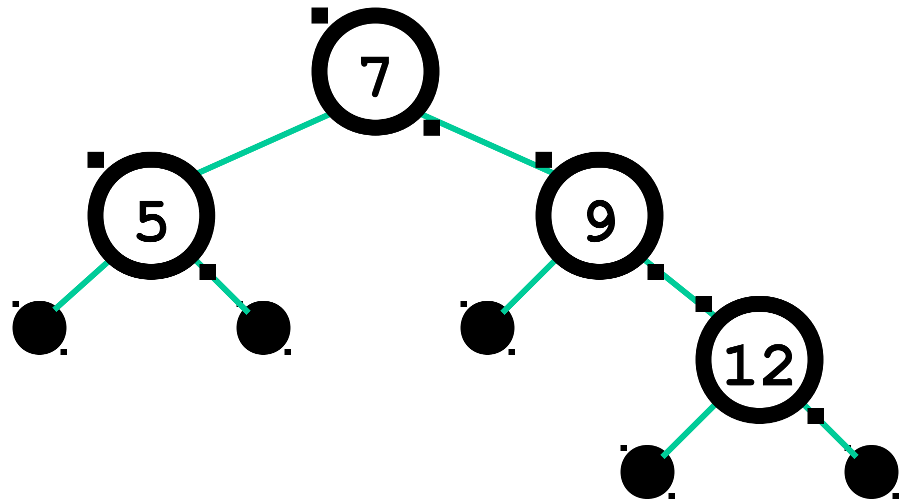
- Black-height of a node:
  - Number of black nodes on any simple path from, but not including, a node x down to a leaf
- A red-black tree with n internal nodes has height at most 2lg(n+1)

Rotations

- Why are rotations necessary in red-black trees?
- How are rotations performed?
- What is the running time of rotation?

Example

- Color this tree
- Insert 8
- Insert 11
- Insert 10



Properties of RB-Trees

1. Every node is either red or black.
2. The root is black.
3. Every leaf is black.
4. If a node is red, both children are black.
5. Every path from node to descendent leaf contain the same number of black nodes.

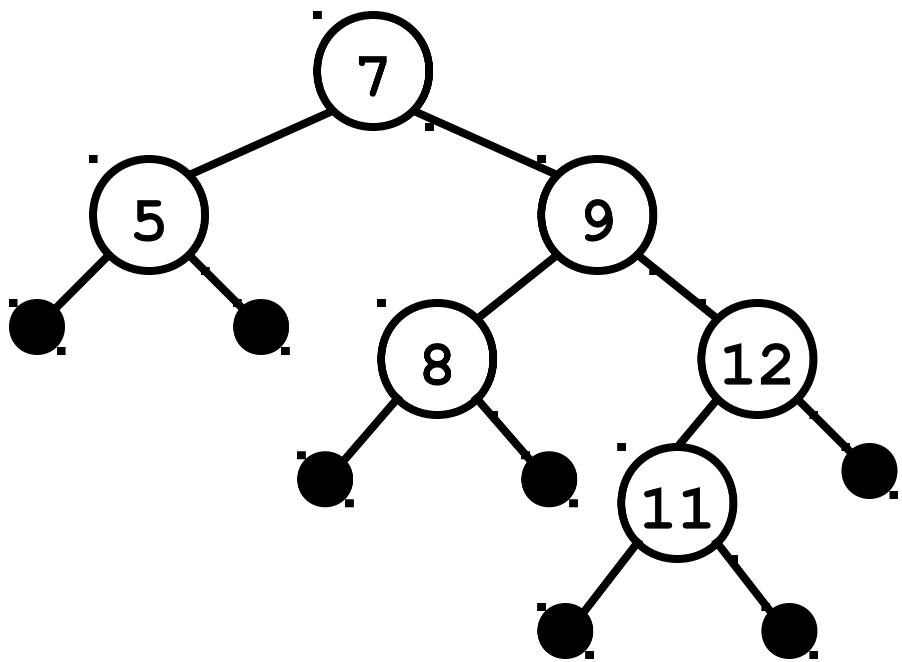
Left-Rotate

LEFT-ROTATE(T, x)

```
y = x.right                 // set y
x.right = y.left            // turn y's left subtree into x's right subtree
if y.left ≠ T.nil
    y.left.p = x
y.p = x.p                   // link x's parent to y
if x.p == T.nil
    T.root = y
elseif x == x.p.left
    x.p.left = y
else x.p.right = y
y.left = x                  // put x on y's left
x.p = y
```

Example

- Rotate left about 9



Inserting into a RB-Tree

- This is regular binary search tree insertion
- Which RB-Tree property could have been violated?

Properties of RB-Trees

1. Every node is either red or black.
2. The root is black.
3. Every leaf is black.
4. If a node is red, both children are black.
5. Every path from node to descendent leaf contain the same number of black nodes.

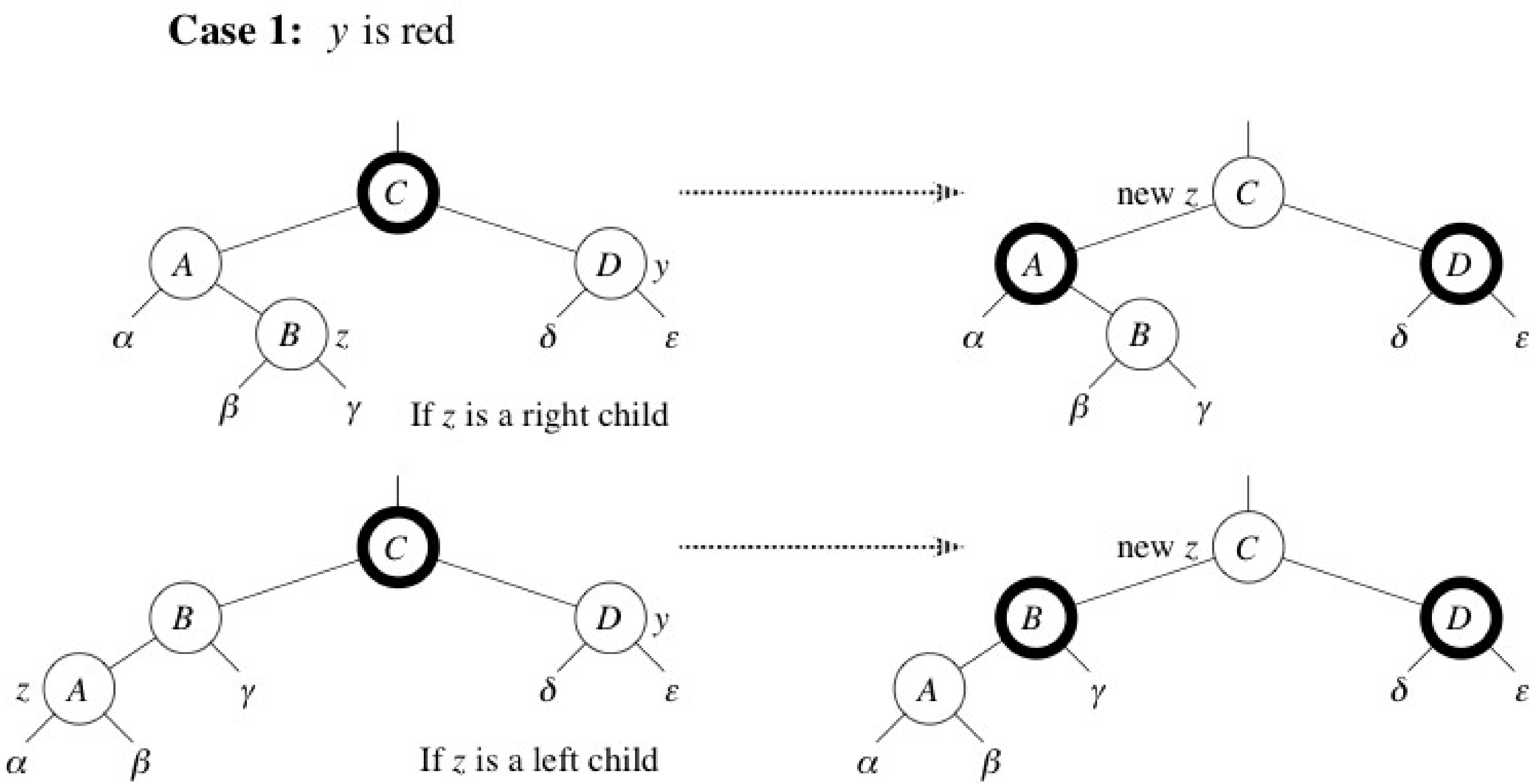
RB-INSERT(T, z)

$y = T.nil$
$x = T.root$
while $x \neq T.nil$
    $y = x$
    if $z.key < x.key$
        $x = x.left$
    else $x = x.right$
$z.p = y$
if $y == T.nil$
    $T.root = z$
elseif $z.key < y.key$
    $y.left = z$
else $y.right = z$
$z.left = T.nil$
$z.right = T.nil$
$z.color =$ RED
RB-INSERT-FIXUP(T, z)

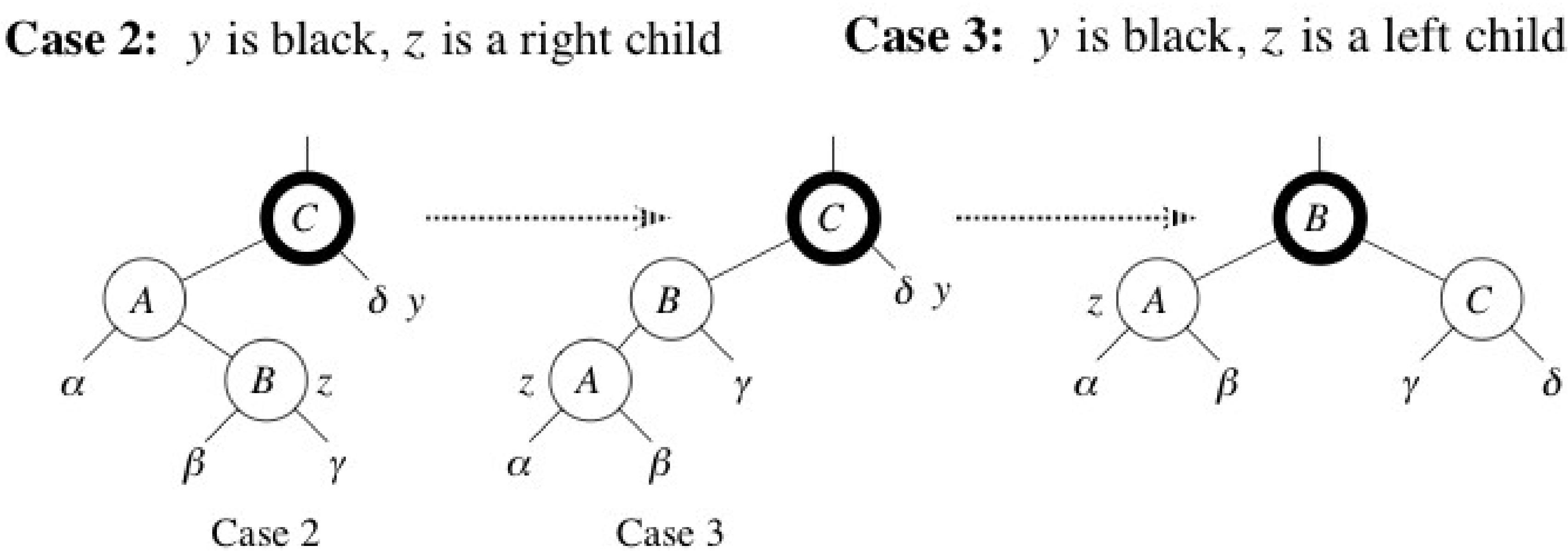
RB-Insert-Fixup

```
RB-INSERT-FIXUP(T, z)
while z.p.color == RED
    if z.p == z.p.p.left
        y = z.p.p.right
        if y.color == RED
            z.p.color = BLACK                                  // case 1
            y.color = BLACK                                    // case 1
            z.p.p.color = RED                                  // case 1
            z = z.p.p                                          // case 1
        else if z == z.p.right
                z = z.p                                        // case 2
                LEFT-ROTATE(T, z)                              // case 2
            z.p.color = BLACK                                  // case 3
            z.p.p.color = RED                                  // case 3
            RIGHT-ROTATE(T, z.p.p)                             // case 3
    else (same as then clause with "right" and "left" exchanged)
T.root.color = BLACK
```

Cases

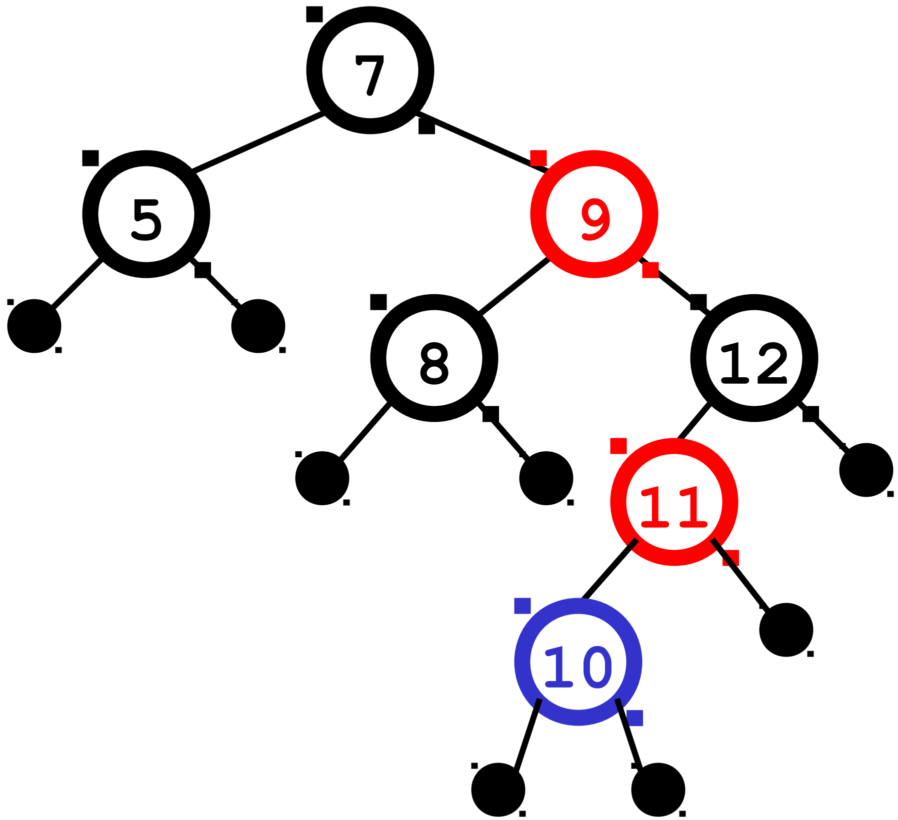

Case 1: y is red
C
A
D
y
α
B
z
δ
ε
β
γ
If z is a right child
new z
C
A
D
α
B
δ
ε
β
γ
C
B
D
y
z
A
γ
δ
ε
α
β
If z is a left child
new z
C
B
D
γ
δ
ε
A
α
β

Cases


Case 2: y is black, z is a right child
Case 3: y is black, z is a left child
C
A
δ y
α
B z
β
γ
Case 2
C
B
δ y
z A
γ
α
β
Case 3
B
z A
C
α
β
γ
δ

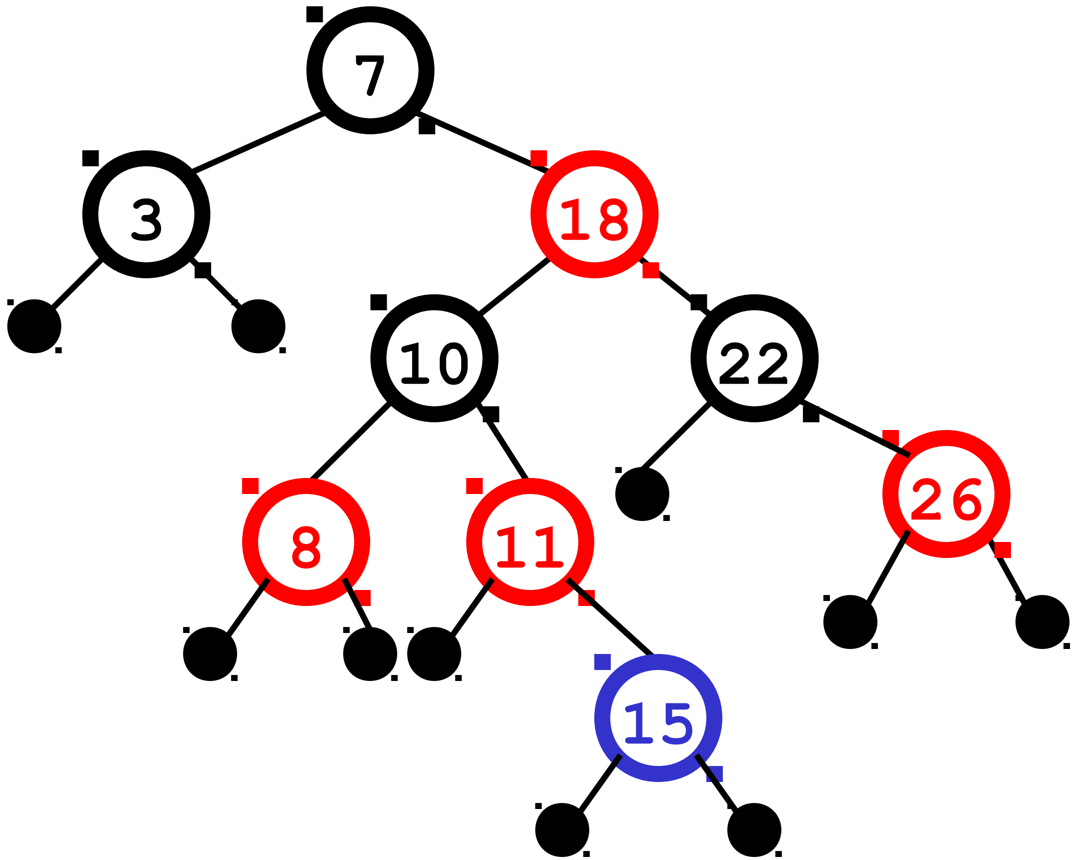
Example

- Insert 10



Example

- Insert 15



Delete BST, p 295 - 298

- BST:
  delete x, 3 cases:
  - x has no children
  - x has 1 child
  - x has 2 children
- helper function *transplant(T, u, v)*
  -

p 295 p 323

```
BST- Transplant(T, u, v)
if u.p == NIL
    T.root = v
elseif u == u.p.left
    u.p.left = v
else u.p.right = v
if v != Nil
    v.p = u.p
```

```
RB-Transplant(T, u, v)
if u.p == T.NIL
    T.root = v
elseif u == u.p.left
    u.p.left = v
else u.p.right = v
v.p = u.p
```

Delete, RB Tree, p 323

- RB-Transplant
- RB-Delete
- RB-DeleteFixup

p324

```
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)
```

p326

```
RB-DELETE-FIXUP(T, x)
  while x ≠ 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
```

Examples

Delete each of these from the original

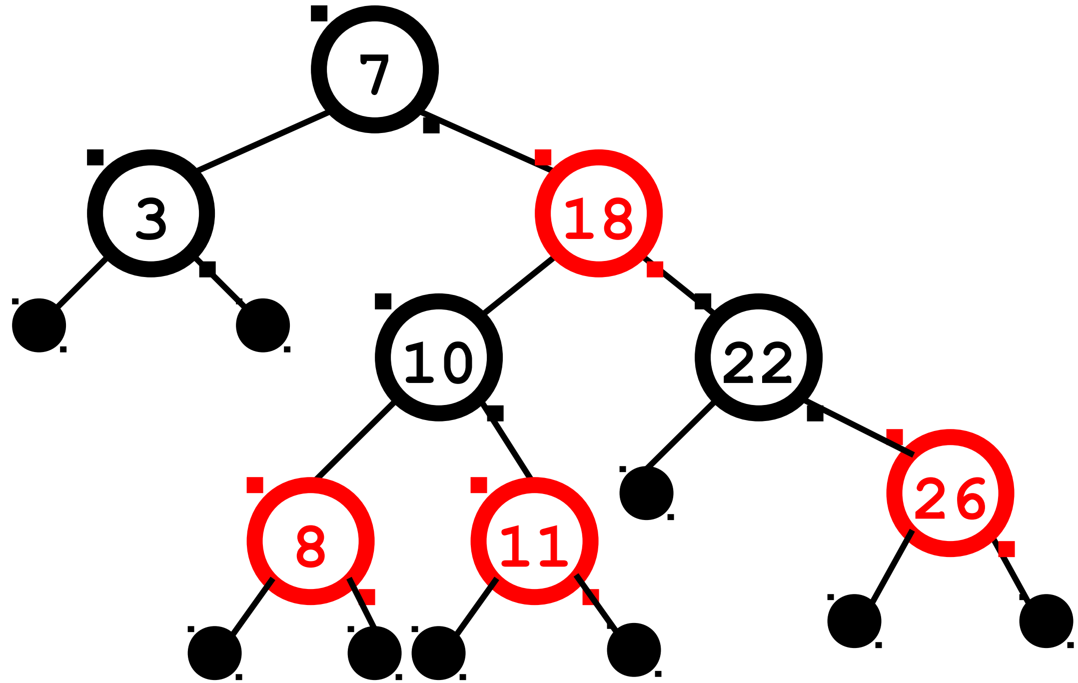
Delete 26

Delete 22

Delete 10

Delete 18

Delete 3



Notes