Lecture 3

Instructor: Subrahmanyam Kalyanasundaram

19th August 2019

Last Class

- BST Delete

Last Class

- BST Delete
- Red-Black Trees
- Today we see, Insert in RB Trees

Red-Black Trees

RBTs have the following properties:

1. All nodes are colored either Red or Black.
2. The root node and the leaf nodes (NIL) are black.
3. Both children of a red node are black.
   No double red.
4. For any node x, all paths from x to the descendant leaves have the same number of black nodes. = Black height(x)

Red-Black Trees

RBTs have the following properties:

1. All nodes are colored either Red or Black.
2. The root node and the leaf nodes (NIL) are black.
3. Both children of a red node are black.
   No double red.
4. For any node x, all paths from x to the descendant leaves have the same number of black nodes. = Black height(x)

Black height of a red black tree is the black height of its root.

A Red-Black Tree supports all procedures of a BST:

- INSERT(*val*) – Inserts *val* into the RBT rooted at *node*.
- SEARCH(*val*) – Returns True of *val* exists in the BST rooted at *node*. False otherwise.
- SUCC(*val*) – Returns the smallest element greater than *val* in the RBT.
- PRED(*val*) – Returns the largest element lesser than *val* in the RBT.
- DELETE(*val*) – Deletes *val* from the RBT.

The procedures in green are implemented exactly like in a BST.

Claim

A red-black tree with black-height β has height at most 2β.
Proof sketch:

- Try to construct the longest possible path with at most β many black nodes.
- Property 4 will force you to color every alternate node black.

Observations

Theorem

If a red-black tree with *n internal* nodes and black height β, then

$$2^{\beta} \leq n + 1 \leq 4^{\beta}.$$

Observations

Theorem

If a red-black tree with *n internal* nodes and black height β, then

$$2^\beta \leq n + 1 \leq 4^\beta.$$

Proof sketch

- Merge each red node with its parent.
- Now each node has 1, 2 or 3 values with 2, 3 or 4 children.
  This is a 2-3-4 tree!

Observations

Theorem

If a red-black tree with *n internal* nodes and black height β, then

$$2^\beta \leq n + 1 \leq 4^\beta.$$

Proof sketch

- Merge each red node with its parent.
- Now each node has 1, 2 or 3 values with 2, 3 or 4 children.
  This is a 2-3-4 tree!
- The above 2-3-4 tree has height β.
- Thus $2^\beta - 1 \leq n \leq 4^\beta - 1$.

Observations

Theorem

A red-black tree with n *internal* nodes has height at most $2\log(n+1)$.

Observations

Theorem

A red-black tree with n *internal* nodes has height at most $2\log(n+1)$.

Proof

- We have seen that $2^{\beta} \leq n+1 \leq 4^{\beta}$.
- That is, $1/2\log(n+1) \leq \beta \leq \log(n+1)$.

Observations

Theorem

A red-black tree with n *internal* nodes has height at most $2\log(n+1)$.

Proof

- We have seen that $2^\beta \leq n+1 \leq 4^\beta$.
- That is, $1/2\log(n+1) \leq \beta \leq \log(n+1)$.
- Use previous claim that height is at most twice the black-height to conclude the Theorem.

INSERT procedure

INSERT(x) – Insert value x into the red-back tree.
High level strategy:

- Create a node X with value x and color red.
- Insert node X just like inserting into a Binary Search Tree.
- Call procedure FIXINSERT at node X.

FixInsert procedure

Which properties might be broken when we insert a new red node?

1. All nodes are colored either Red or Black.
2. The root node is black.
3. The leaf nodes (NIL) are black.
4. Both children of a red node are black.
5. For any node, all paths from the node to the descendant leaves have the same number of black nodes.

FixInsert procedure

Only properties 2 and 4 could be broken after inserting a red node:

1. All nodes are colored either Red or Black.
2. The root node is black.
3. The leaf nodes (NIL) are black.
4. Both children of a red node are black.
5. For any node, all paths from the node to the descendant leaves have the same number of black nodes.

FixInsert procedure

Only properties 2 and 4 could be broken after inserting a red node:

1. All nodes are colored either Red or Black.
2. The root node is black.
3. The leaf nodes (NIL) are black.
4. Both children of a red node are black.
5. For any node, all paths from the node to the descendant leaves have the same number of black nodes.

FixInsert - Fixing property 2

Property 2

The root node is black.

Some invariants when FixInsert is called on a node Z:

- Z is colored Red.
- If Property 2 is violated, then node Z itself is the root.

FixInsert - Fixing property 2

Property 2

The root node is black.

Some invariants when FixInsert is called on a node Z:

- Z is colored Red.
- If Property 2 is violated, then node Z itself is the root.

Resolution: Simply color Z black.

FixInsert - Fixing Property 4

Property 4

A red node has black children.

FixInsert - Fixing Property 4

Property 4

A red node has black children.

Some invariants when FixInsert is called on a node Z:

- Z is colored Red.
- If Property 4 is violated, it is violated only by the node Z and its parent.

FixInsert - Fixing Property 4

Property 4

A red node has black children.

Some invariants when FixInsert is called on a node Z:

- Z is colored Red.
- If Property 4 is violated, it is violated only by the node Z and its parent.

There are three cases when FixInsert is called on a node Z.

Cases

- Case 1: Uncle of Z is Red.

Cases

- Case 1: Uncle of Z is Red.
- Case 2: Uncle of Z is black and Z is a right child of a left child.

Cases

- Case 1: Uncle of Z is Red.
- Case 2: Uncle of Z is black and Z is a right child of a left child.
- Case 3: Uncle of Z is black and Z is a left child of a left child.

Cases

- Case 1: Uncle of Z is Red.
- Case 2: Uncle of Z is black and Z is a right child of a left child.
- Case 3: Uncle of Z is black and Z is a left child of a left child.
- Other cases follow by symmetry.

Case 1

Case 1

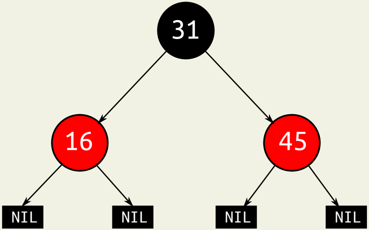
Uncle of Z is Red.

Resolution:

- Recolor parent, uncle and grandparent.
- Call FixInsert(grandparent)

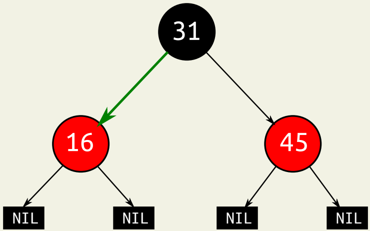
Example 1

Insert 9 to the following RBT:



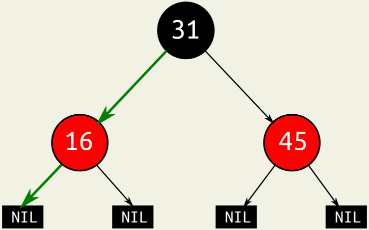
Example 1

Find the position where 9 should be inserted



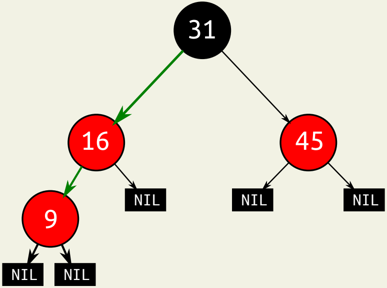
Example 1

Find the position where 9 should be inserted



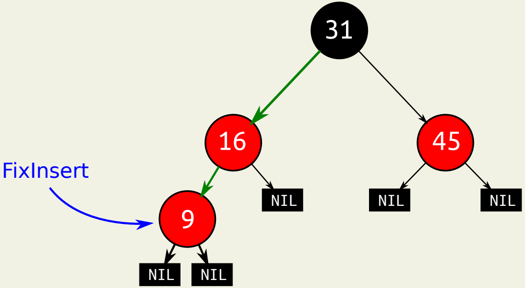
Example 1

Insert 9 as a new node with color red



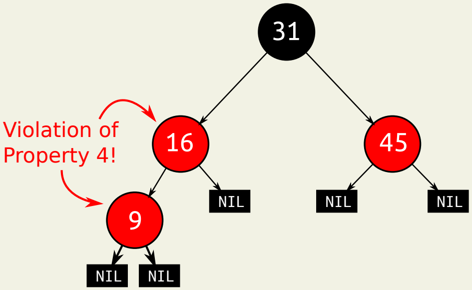
Example 1

Call FixInsert at the inserted location.



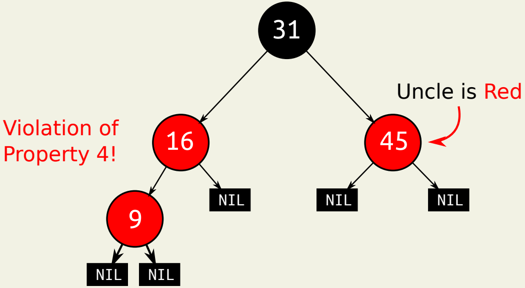
Example 1

Property 4 is violated. Check color of the uncle to determine case.



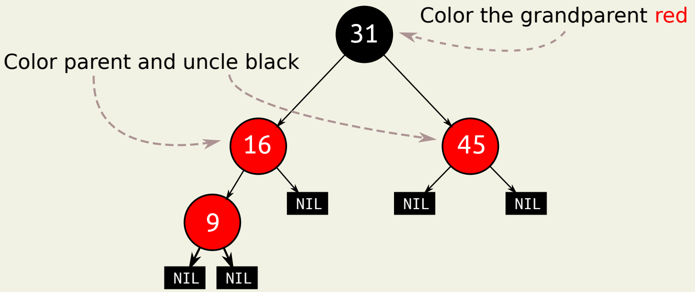
Example 1

We are in Case 1.



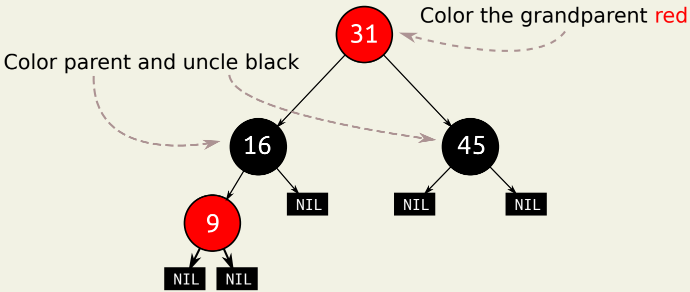
Example 1

Case 1 is resolved by recoloring.



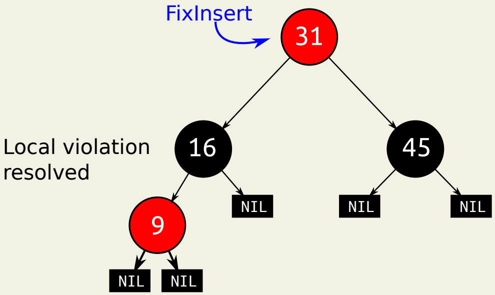
Example 1

Case 1 is resolved by recoloring.



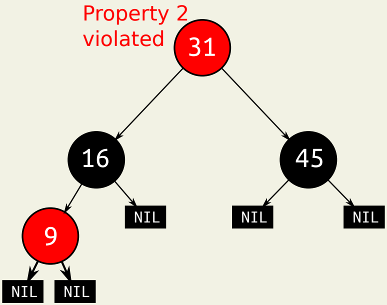
Example 1

Now call FIXINSERT on the grandparent.



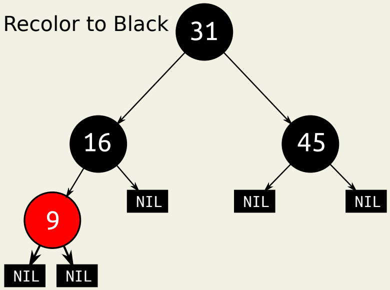
Example 1

Root node is not black.



Example 1

Simply recolor root to black.



Case 3

Case 3

Uncle of Z is Black and Z is left child of a left child.

Resolution:

- Recolor parent and grandparent.
- Rotate right at grandparent.

Case 3

Case 3

Uncle of Z is Black and Z is left child of a left child.

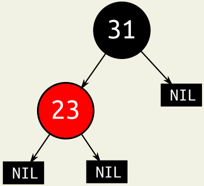
Resolution:

- Recolor parent and grandparent.
- Rotate right at grandparent.

Symmetric case: Z is right child of a right child.

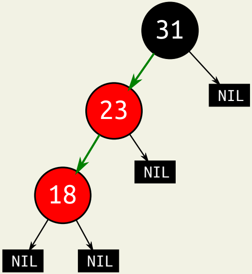
Example 2

Want to insert 18 into this RBT.



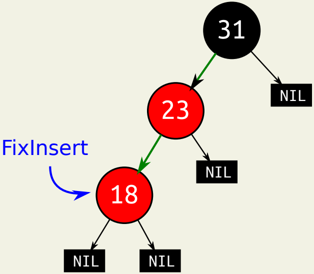
Example 2

Insert 18 as a red node according to BST property.



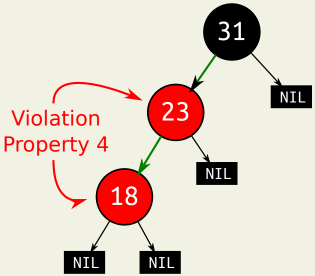
Example 2

Call FixInsert on the new node.



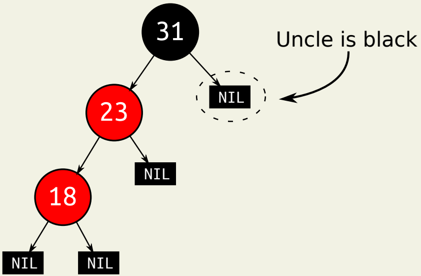
Example 2

FIXINSERT has to fix property 4.



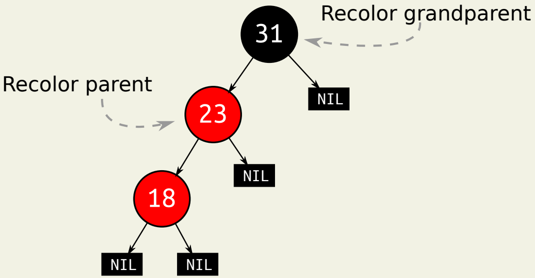
Example 2

Determine the case by looking at uncle.



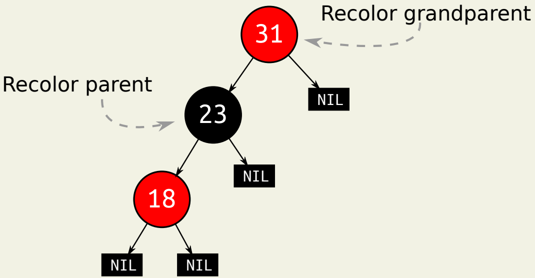
Example 2

Node 18 and its parent 23 are both left children. This is case 3.



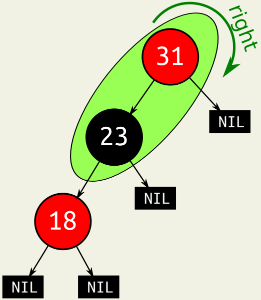
Example 2

Recolor parent to black and grandparent to red.



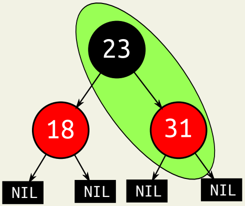
Example 2

Rotate right at grandparent.



Example 2

The resulting tree has no violations.



Case 2

Case 2

Uncle of Z is Black and Z is right child of a left child.

Resolution:

- Assign parent to Z.
- Rotate left at Z.
- Call FixInsert at Z.

The above procedure results in case 3.

Case 2

Uncle of Z is Black and Z is right child of a left child.

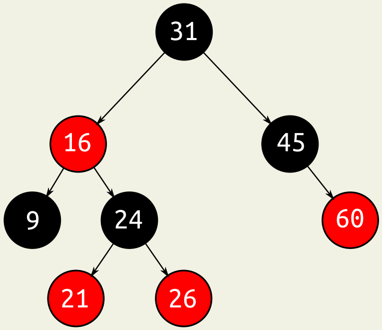
Resolution:

- Assign parent to Z.
- Rotate left at Z.
- Call FixInsert at Z.

The above procedure results in case 3.
Symmetric case: Z is left child of a right child.

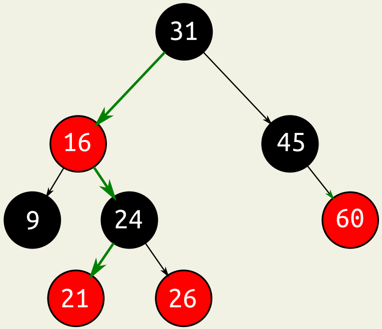
Example 3

Want to insert 18 into the following:



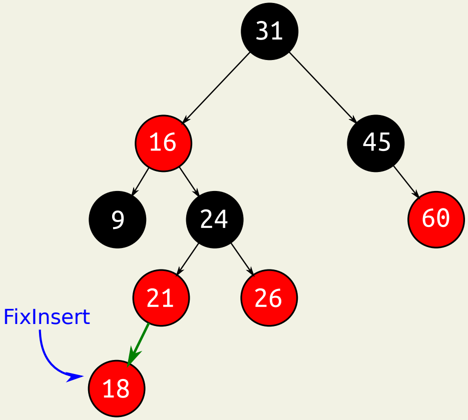
Example 3

Find the position for 18:



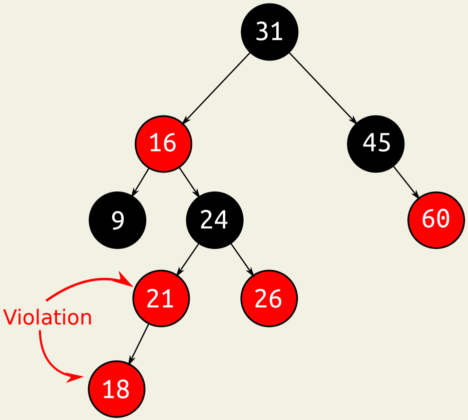
Example 3

Insert a new node with value 18 and color red. Call FixInsert at the inserted location.



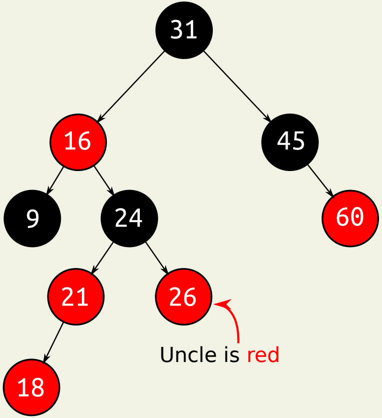
Example 3

Property 4 is violated. Check color of uncle to determine the case.



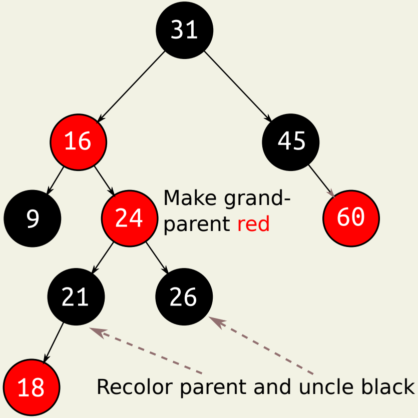
Example 3

Uncle is red. So we are in case 1.



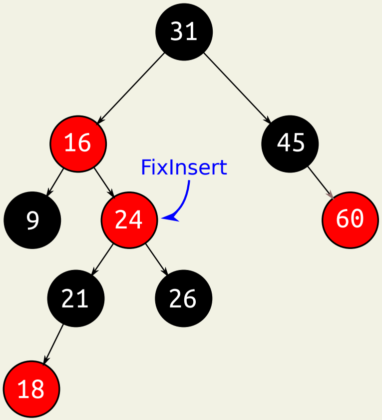
Example 3

Recolor as done earlier.



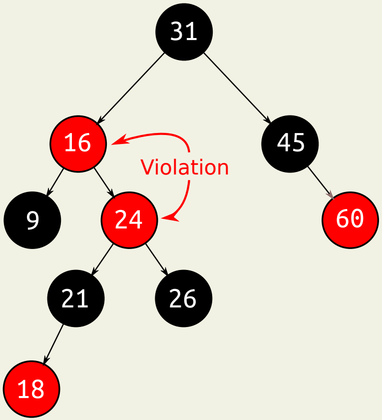
Example 3

Call FixInsert on grandparent.



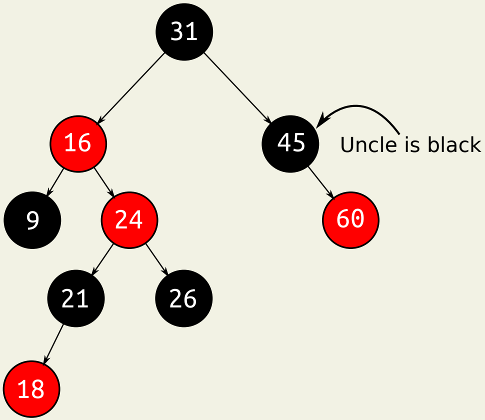
Example 3

Property 4 does not hold. Check color of Uncle to determine case.



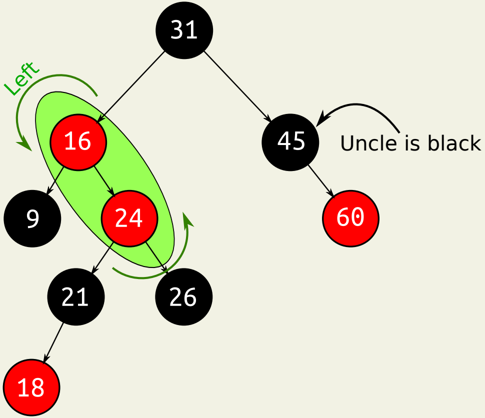
Example 3

Uncle is black and 24 is right child of a left child. So we are in case 2.



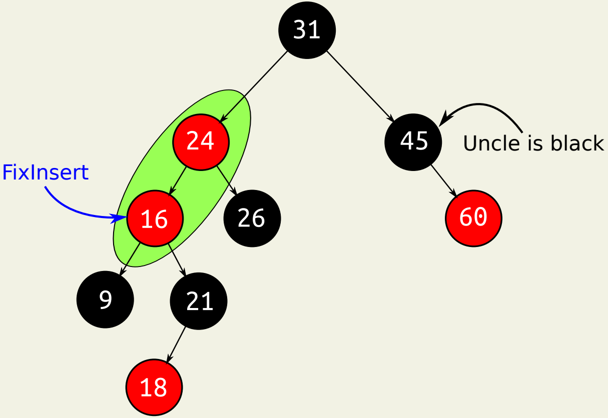
Example 3

Set Z as node with 16. Rotate left at Z and call FixInsert on Z.



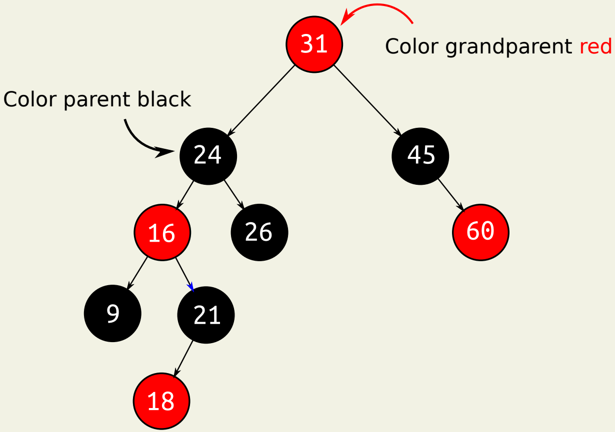
Example 3

Now, Uncle is black and 16 is left child of a left child. This is case 3.



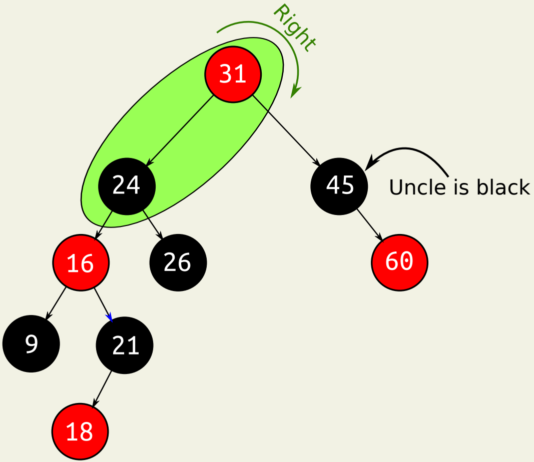
Example 3

Recolor granparent red, parent black.



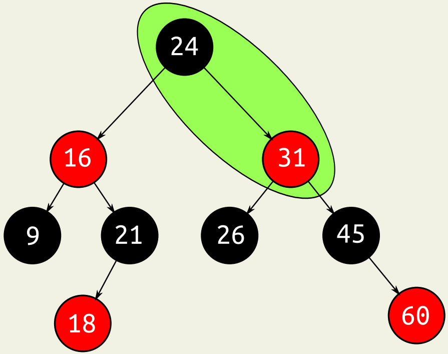
Example 3

Right rotate at grandparent.



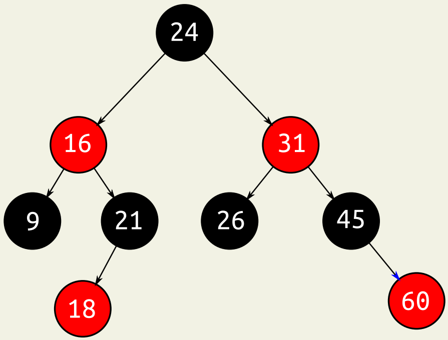
Example 3

Right rotate at grandparent.



Example 3

Done!



FixInsert pseudocode

Algorithm 1 FixInsert called on node Z

```
while color(parent(Z)) = red do
    U ← Uncle(Z)
    if parent(Z) is the left child of the grandparent then
        if color(U) = red then
            Recolor parent, uncle and grandparent.
            Z ← grandparent(Z).
        else
            if Z is the right child then
                Z ← parent(Z); Left rotate at (Z)
            end if
            Recolor parent and grandparent.
            Right rotate at grandparent(Z).
        end if
    end if
end while
color(root)← black.
```