Algorithm Analysis and Design

Advanced Data Structure
(Red Black Tree)

(Deletion)

Lecture -30-32

Overview

- A variation of binary search trees.
- Balanced: height is O(lg n), where n is the number of nodes.
- Operations will take O(lg n) time in the worst case.

Red Black Tree (Deletion)

Start by doing regular binary-search-tree deletion:

Red Black Tree (Deletion)

```
RB-DELETE(T, z)
if left[z] = nil[T] or right[z] = nil[T ]
   then y ← z
   else y ← TREE-SUCCESSOR(z)
if left[y] ≠ nil[T ]
    then x ← left[y]
    else x ← right[y]
p[x] ← p[y]
if p[y] = nil[T ]
     then root[T ] ← x
     else if y = left[p[y]]
               then left[p[y]] ← x
               else right[p[y]] ← x
if y ≠ z
      then key[z] ← key[y]
               copy y's satellite data into z
if color[y] = BLACK
       then RB-DELETE-FIXUP(T, x)
return y
```

Red Black Tree (Deletion)

```
RB-DELETE(T, z)
if left[z] = nil[T] or right[z] = nil[T ]
   then y ← z
   else y ← TREE-SUCCESSOR(z)
if left[y] ≠ nil[T ]
    then x ← left[y]
    else x ← right[y]
p[x] ← p[y]
if p[y] = nil[T ]
     then root[T ] ← x
     else if y = left[p[y]]
               then left[p[y]] ← x
               else right[p[y]] ← x
if y ≠ z
      then key[z] ← key[y]
               copy y's satellite data into z
if color[y] = BLACK
       then RB-DELETE-FIXUP(T, x)
return y
```

- 'y' is the node that was actually deleted(i.e. Successor of 'z').
- 'x' is either
  - *y's* sole non-sentinel child before y was deleted, or
  - the sentinel, if y had no children.
- In both cases, p[x] is now the node that was previously *y's* parent.

Red Black Tree (Deletion)

Now Check

- If y is black, we could have violations of red-black properties:
- Which property might be violated?

P1. Every node is either red or black ?

OK

P2. The root is always black ?

If y is the root and x is red, then the root has become red.

P3. Every leaf (nil[T]) is black ?

OK

P4. If a node is red, then both its children are black ?

Violation if p[y] and x are both red.

Red Black Tree (Deletion)

Now Check

P5. For each node, all paths from the node to descendant leaves contain the same number of black nodes ?

- Any path containing y now has 1 fewer black node.
- Correct by giving x an "extra black".
- Add 1 to count of black nodes on paths containing x.
- Now property 5 is OK, but property 1 is not.
- x is either doubly black (if color[x] = BLACK) or red & black (if color[x] = RED).
- The attribute color[x] is still either RED or BLACK. No new values for color attribute.
- In other words, the extra blackness on a node is by virtue of x pointing to the node.

Red Black Tree (Deletion)

Basic Idea for Deletion

Move the extra black up the tree until

- x points to a red & black node ⇒turn it into a black node,
- x points to the root ⇒just remove the extra black, or
- Do certain rotations (i.e. LL or RR) and recoloring the node and finished the deletion by maintain the Red-black tree property.

Red Black Tree (Deletion)

The basic idea for deletion was executed by the help of RB-DELETE-FIXUP function (i.e. Remove the violations by calling RB-DELETE-FIXUP) .

This function is executed until $x \neq root$ and $color[x] = Black$

For this first find the sibling of 'x' as 'w' and 'w' can not be NIL

There are 8 number of cases:

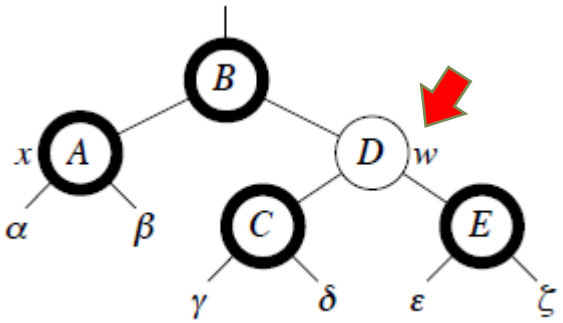
- 4 of which are symmetric to the other 4 (like RB-INSERT-FIXUP()).
- We look at cases in which 'x' is the left child of it's parents.

Red Black Tree (Deletion)

- Case 1:
  - When 'w'(sibling of 'x')is red and w=right[p[x]]

Action to be taken

- Change the color[w]=Black.
- Change the color[p[x]]=Red.
- LEFT_ROTATE(T,p[x])
- Move w=right[p[x]]
- Go immediately to case 2, 3, or 4.



[Nodes with bold outline indicate black nodes and light outline indicate red nodes]

Red Black Tree (Deletion)

- Case 1:
  - When 'w'(sibling of 'x')is red and w=right[p[x]]

Action to be taken

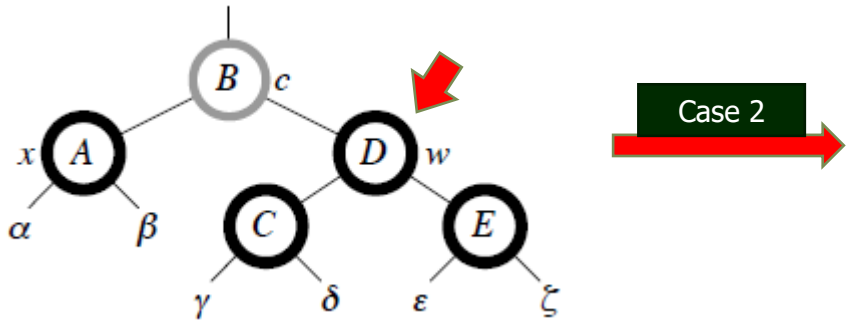
- Change the color[w]=Black.
- Change the color[p[x]]=Red.
- LEFT_ROTATE(T,p[x])
- Move w=right[p[x]]
- Go immediately to case 2, 3, or 4.

[Nodes with bold outline indicate black nodes and light outline indicate red nodes]

Red Black Tree (Deletion)

- Case 2:
  - *When 'w'(sibling of 'x')is black and both the children of of 'w' is also black.*

  Action to be taken
    - Change color[w]=Red.
    - Move x=p[x]

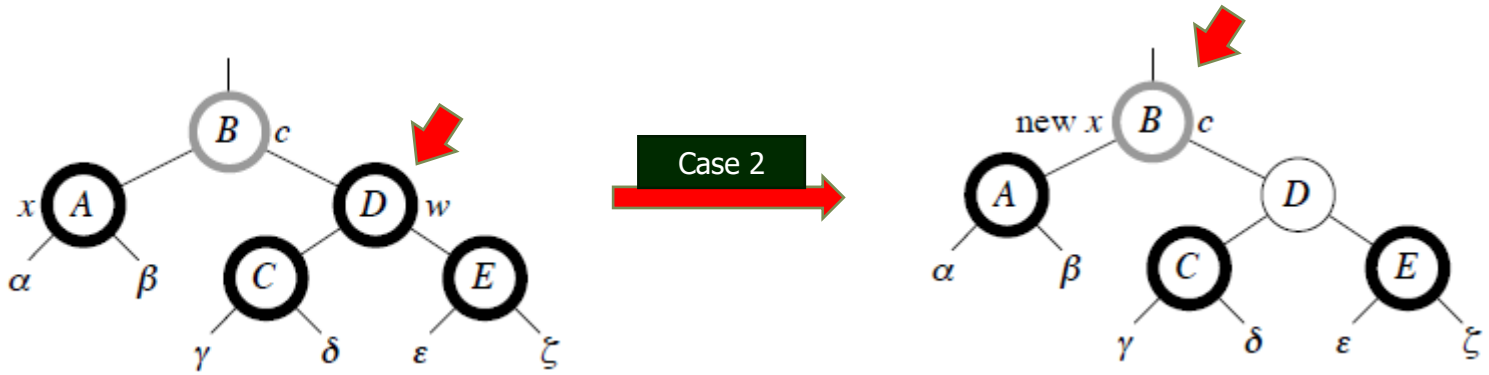


[Nodes with bold outline indicate black nodes and light outline indicate red nodes]
[Node with gray outline mark 'c' represent unknown color (i.e. may be Red or Black)]

Red Black Tree (Deletion)

- Case 2:
  - *When 'w'(sibling of 'x')is black and both the children of of 'w' is also black.*

  Action to be taken
    - Change color[w]=Red.
    - Move x=p[x]



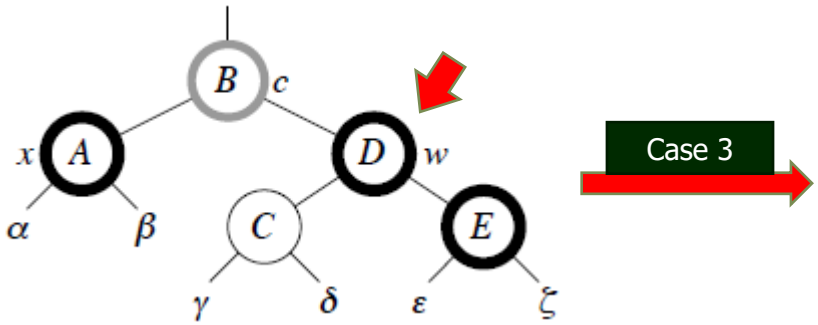
[Nodes with bold outline indicate black nodes and light outline indicate red nodes]
[Node with gray outline mark 'c' represent unknown color (i.e. may be Red or Black)]

Red Black Tree (Deletion)

- Case 3:
  - *When 'w'(sibling of 'x')is black and w's left child is red and w's right child is black.*

Action to be taken

- Color[left[w]]=Black
- Color[w]=Red
- RIGHT_ROTATE(T, w)
- Move w=right[p[x]]



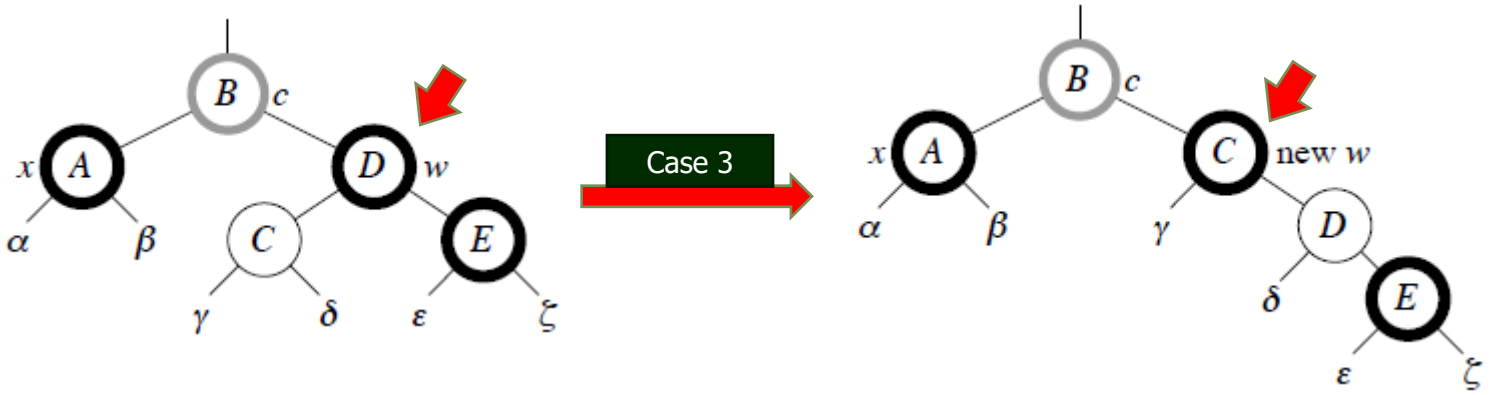
[Nodes with bold outline indicate black nodes and light outline indicate red nodes]
[Node with gray outline mark 'c' represent unknown color (i.e. may be Red or Black)]

Red Black Tree (Deletion)

- Case 3:
  - *When 'w'(sibling of 'x')is black and w's left child is red and w's right child is black.*

Action to be taken

- Color[left[w]]=Black
- Color[w]=Red
- RIGHT_ROTATE(T, w)
- Move w=right[p[x]]



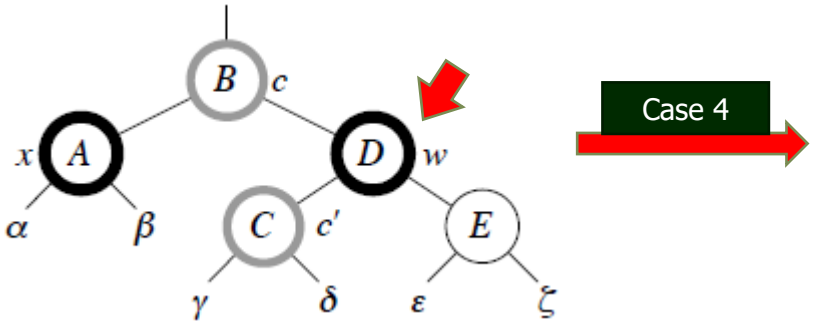
[Nodes with bold outline indicate black nodes and light outline indicate red nodes]
[Node with gray outline mark 'c' represent unknown color (i.e. may be Red or Black)]

Red Black Tree (Deletion)

- Case 4:
  - *When 'w'(sibling of 'x')is black and w's left child is black and w's right child is red.*

Action to be taken

- Color[w]=color[p[x]]
- Color[p[x]]=Black
- Color[right[w]]=Black
- LEFT_ROTATE(T, p[x])
- x=root[T]



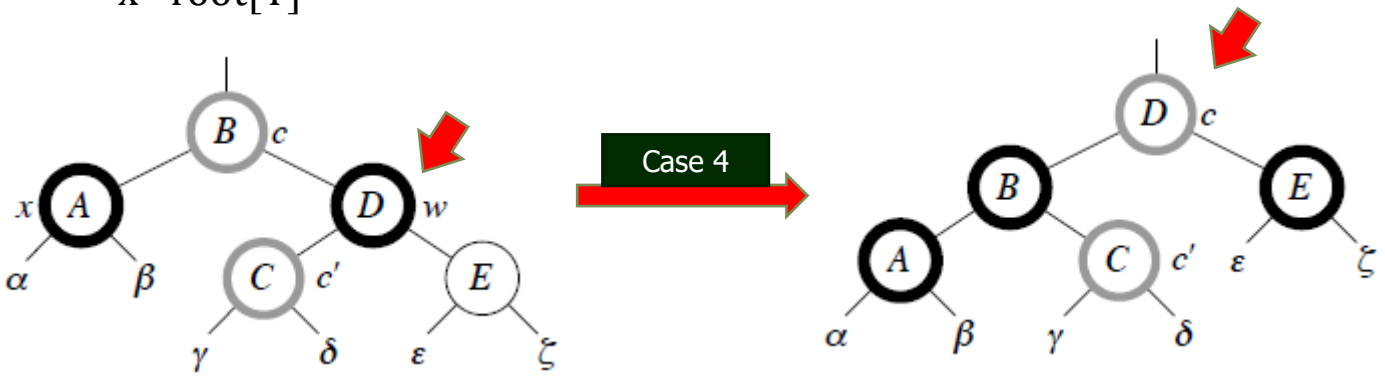
[Nodes with bold outline indicate black nodes and light outline indicate red nodes]
[Node with gray outline mark 'c' represent unknown color (i.e. may be Red or Black)]

Red Black Tree (Deletion)

- Case 4:
  - *When 'w'(sibling of 'x')is black and w's left child is black and w's right child is red.*

Action to be taken

- Color[w]=color[p[x]]
- Color[p[x]]=Black
- Color[right[w]]=Black
- LEFT_ROTATE(T, p[x])
- x=root[T]



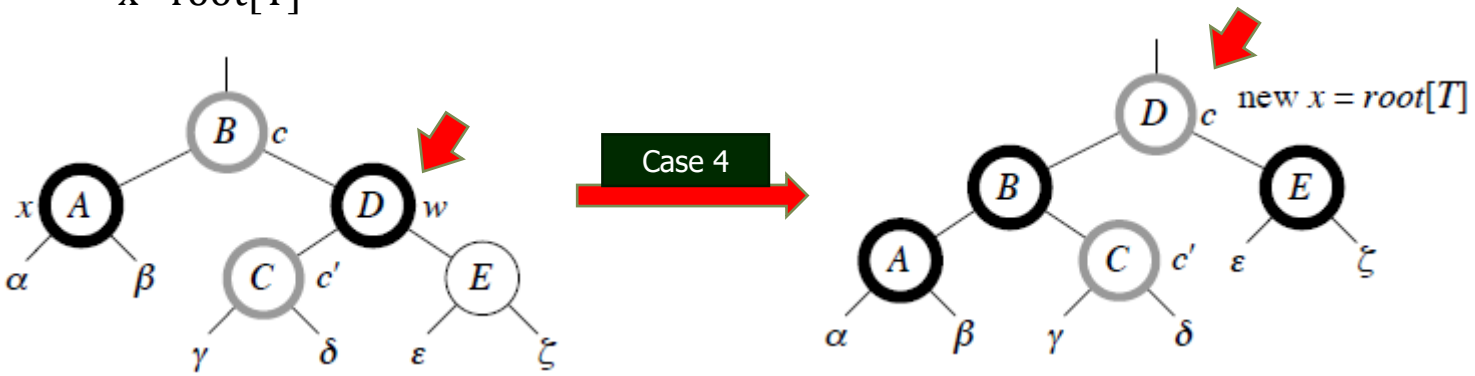
[Nodes with bold outline indicate black nodes and light outline indicate red nodes]
[Node with gray outline mark 'c' represent unknown color (i.e. may be Red or Black)]

Red Black Tree (Deletion)

- Case 4:
  - *When 'w'(sibling of 'x')is black and w's left child is black and w's right child is red.*

Action to be taken

- Color[w]=color[p[x]]
- Color[p[x]]=Black
- Color[right[w]]=Black
- LEFT_ROTATE(T, p[x])
- x=root[T]



[Nodes with bold outline indicate black nodes and light outline indicate red nodes]
[Node with gray outline mark 'c' represent unknown color (i.e. may be Red or Black)]

Red Black Tree (Deletion)

```
RB-DELETE-FIXUP(T, x)
while x ≠ root[T ] and color[x] = BLACK
      do if x = left[p[x]]
            then w ← right[p[x]]
                 if color[w] = RED
                    then color[w] ← BLACK                          ✂ Case 1
                         color[p[x]] ← RED                         ✂ Case 1
                         LEFT-ROTATE(T, p[x])                      ✂ Case 1
                         w ← right[p[x]]                           ✂ Case 1
                 if color[left[w]] = BLACK and color[right[w]] = BLACK
                    then color[w] ← RED                            ✂ Case 2
                         x ← p[x]                                  ✂ Case 2
                    else if color[right[w]] = BLACK
                            then color[left[w]] ← BLACK            ✂ Case 3
                                 color[w] ← RED                    ✂ Case 3
                                 RIGHT-ROTATE(T, w)                ✂ Case 3
                                 w ← right[p[x]]                   ✂ Case 3
                         color[w] ← color[p[x]]                    ✂ Case 4
                         color[p[x]] ← BLACK                       ✂ Case 4
                         color[right[w]] ← BLACK                   ✂ Case 4
                         LEFT-ROTATE(T, p[x])                      ✂ Case 4
                         x ← root[T ]                              ✂ Case 4
         else (same as then clause with "right" and "left" exchanged)
   color[x] ← BLACK
```

Red Black Tree

Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Red Black Tree

Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -50

Red Black Tree

Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -50

Red Black Tree

No Fixup required

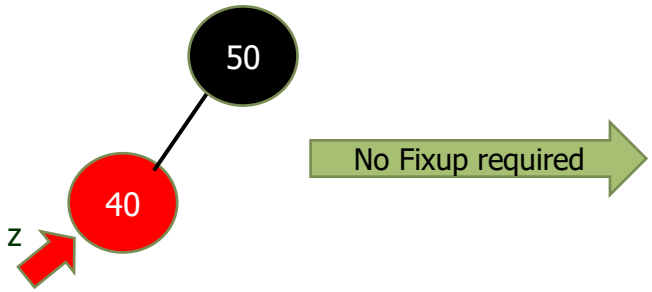
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -40



Red Black Tree

No Fixup required

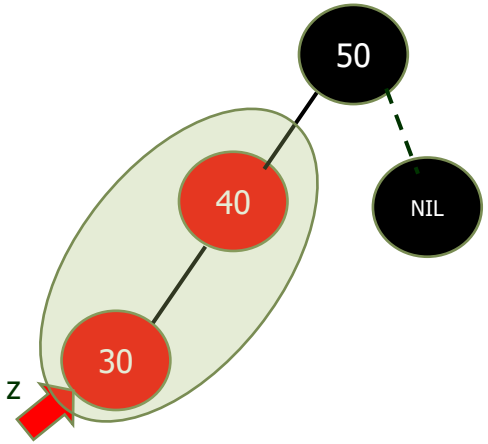
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -30



Red Black Tree

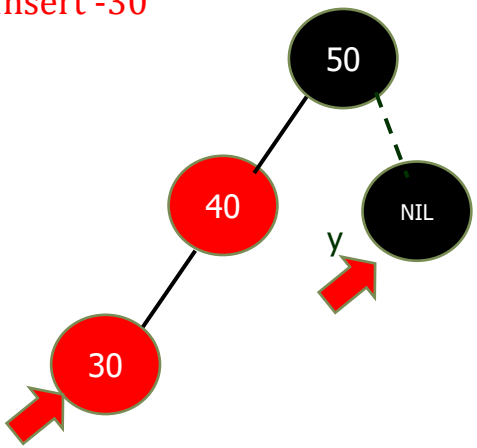
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

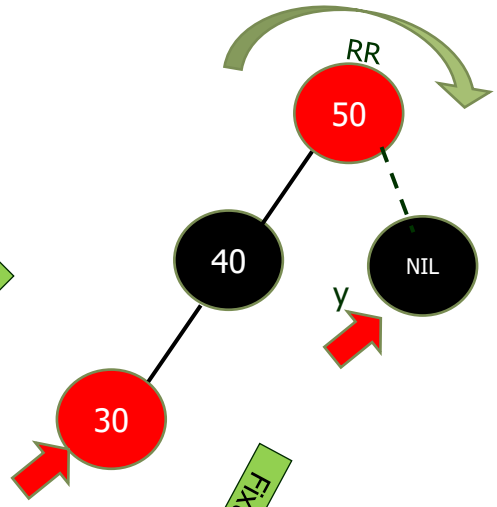
and then perform delete 40, 20, 30, 45, & 5

Insert -30

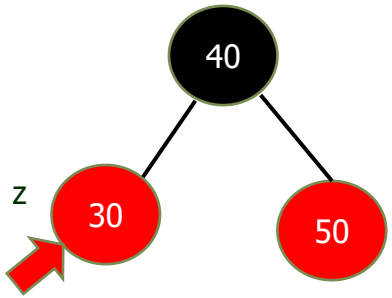


Case 3

Fixup Apply



Fixup Apply After RR



Red Black Tree

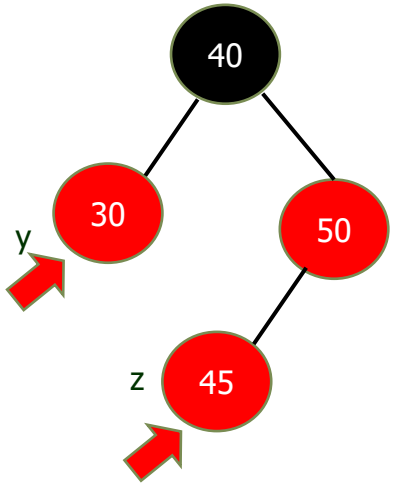
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -45



Red Black Tree

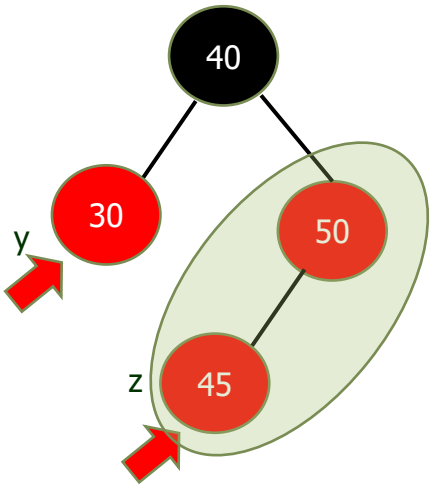
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -45



Red Black Tree

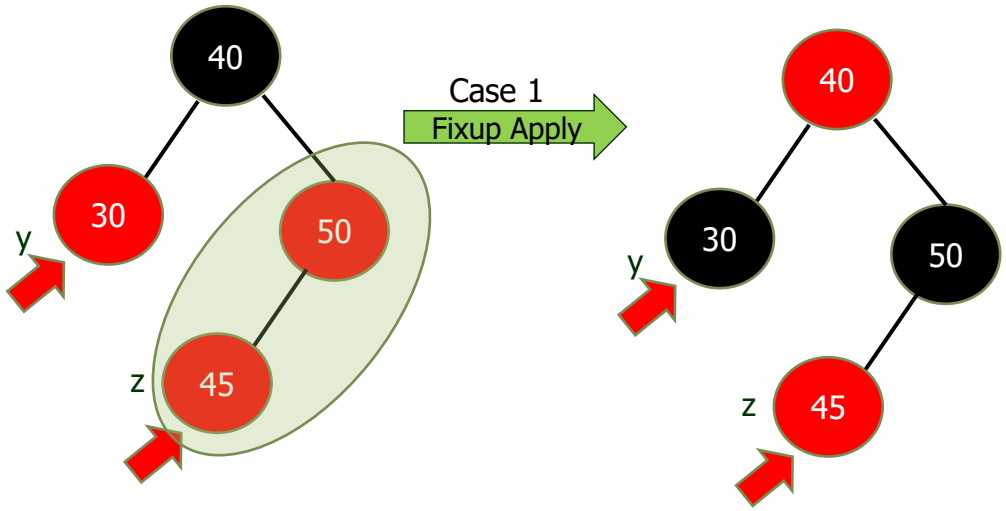
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -45



Red Black Tree

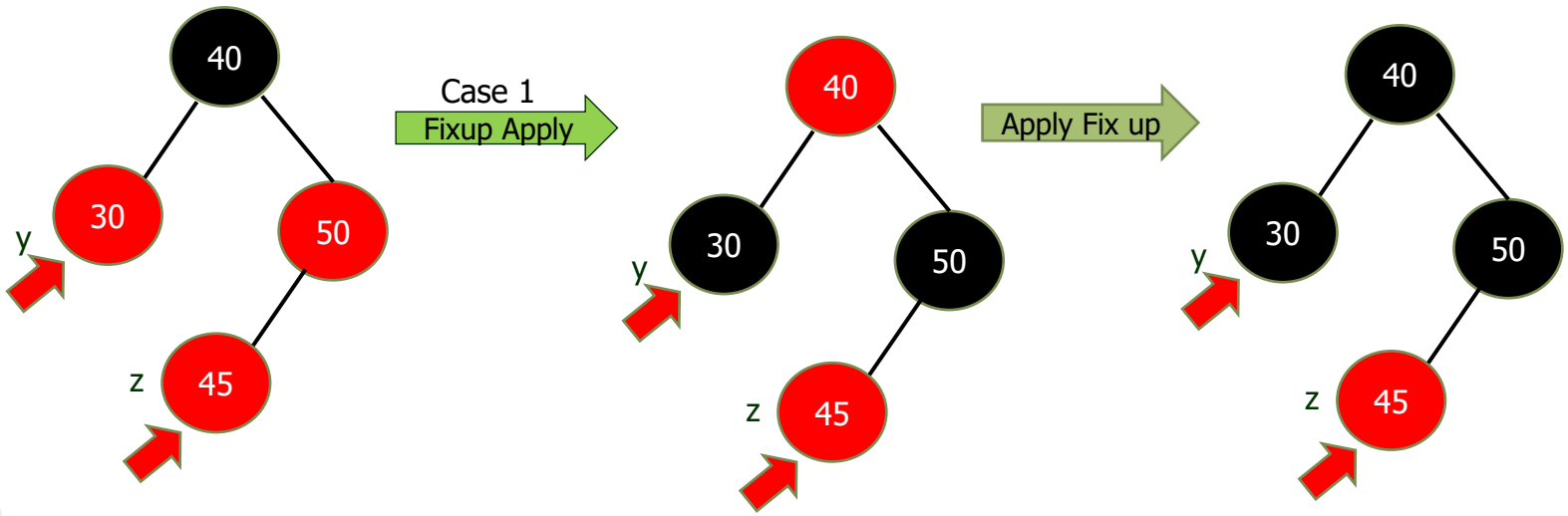
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -45



Red Black Tree

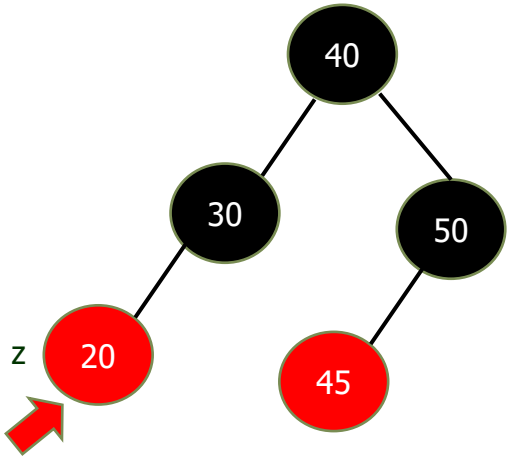
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -20



Red Black Tree

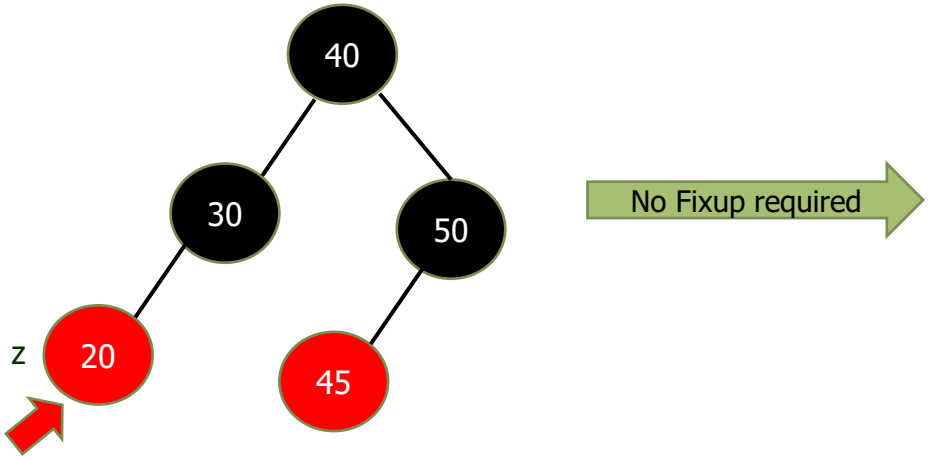
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -20



Red Black Tree

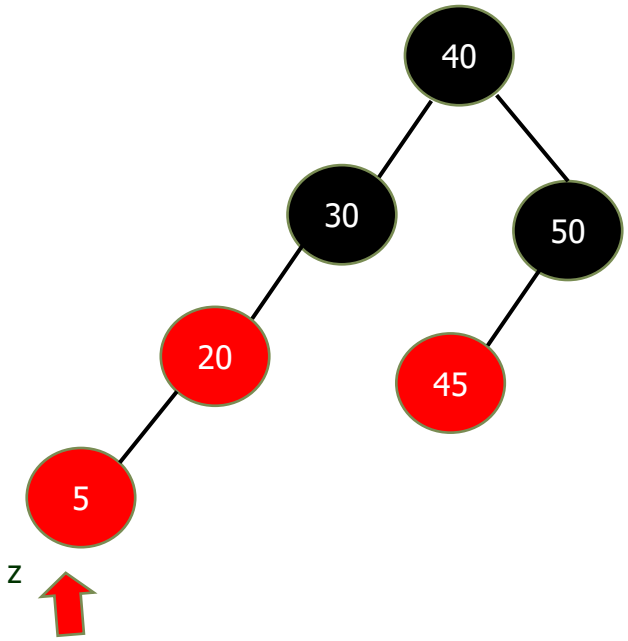
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -5



Red Black Tree

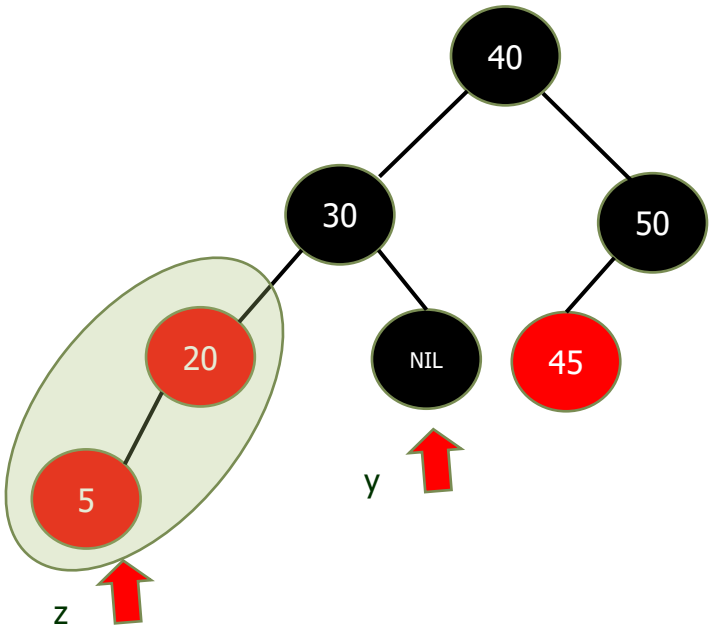
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -5



Red Black Tree

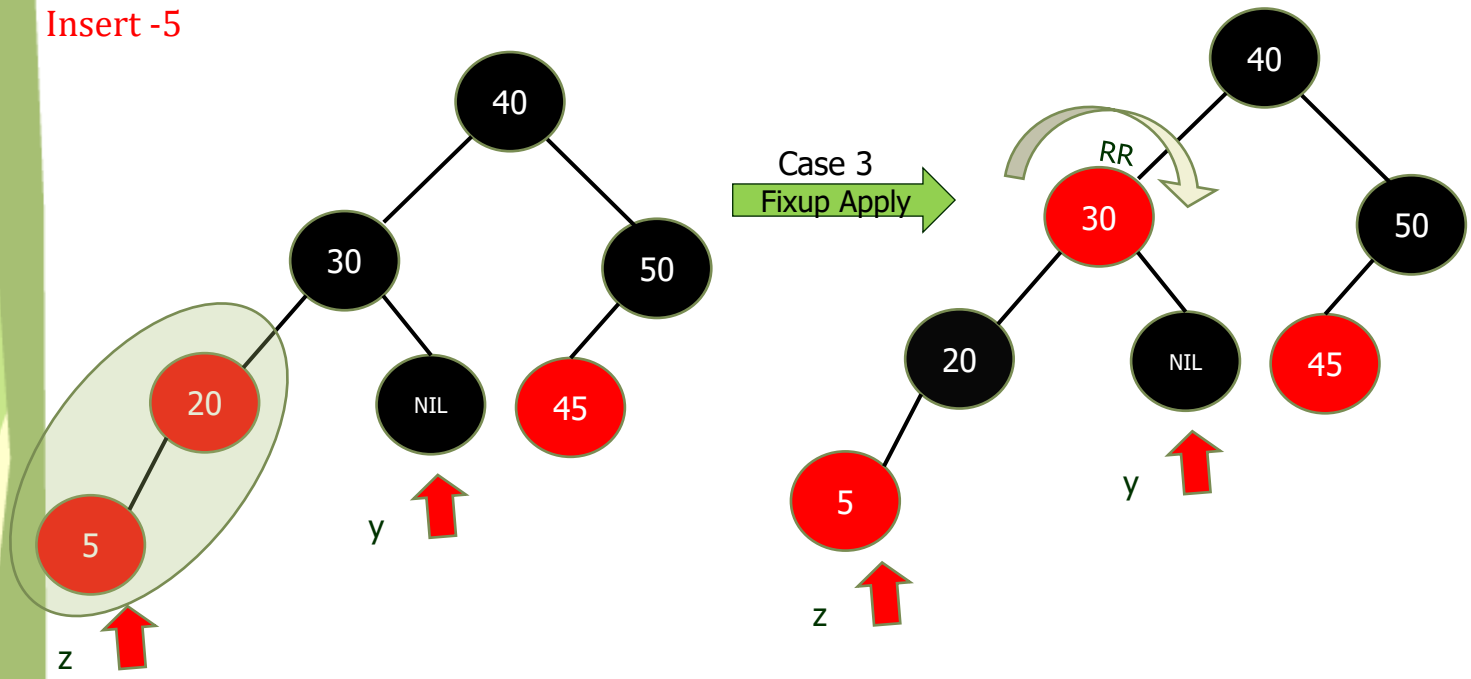
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -5



Red Black Tree

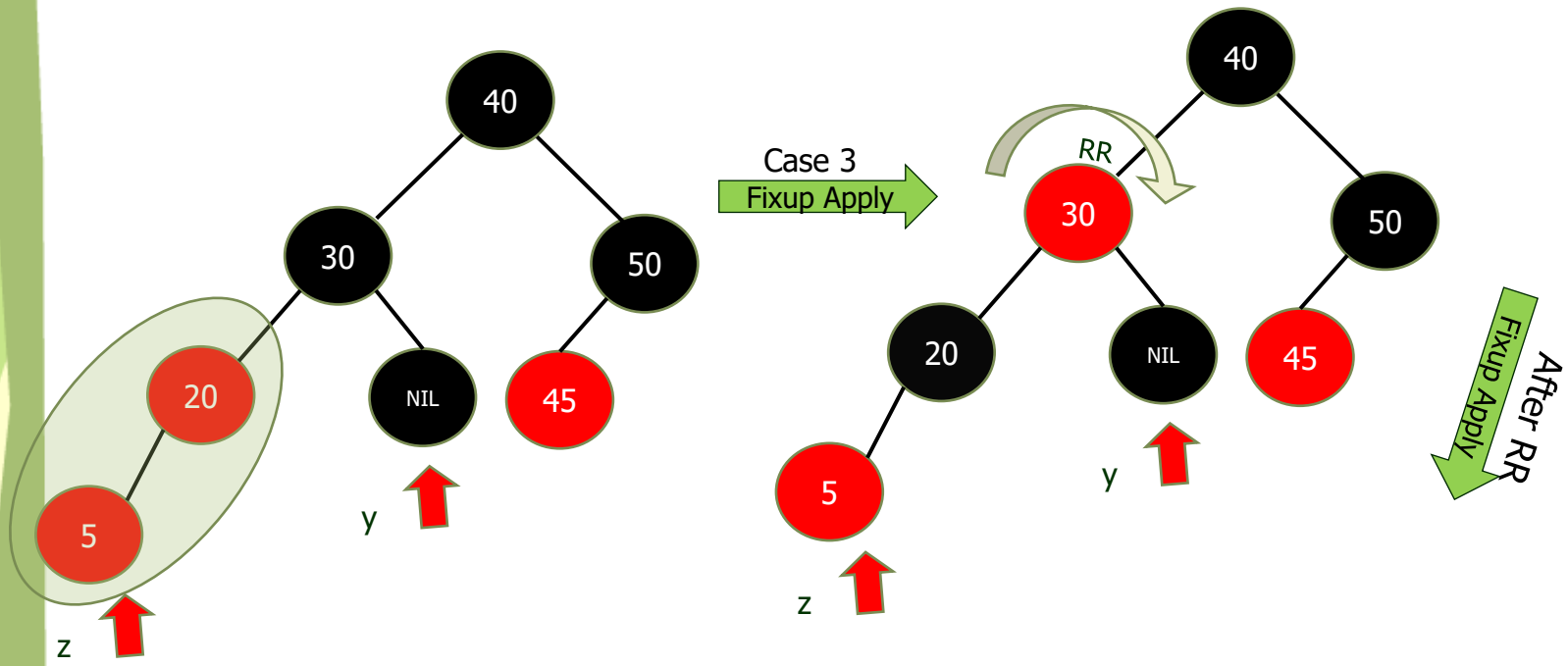
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -5



Red Black Tree

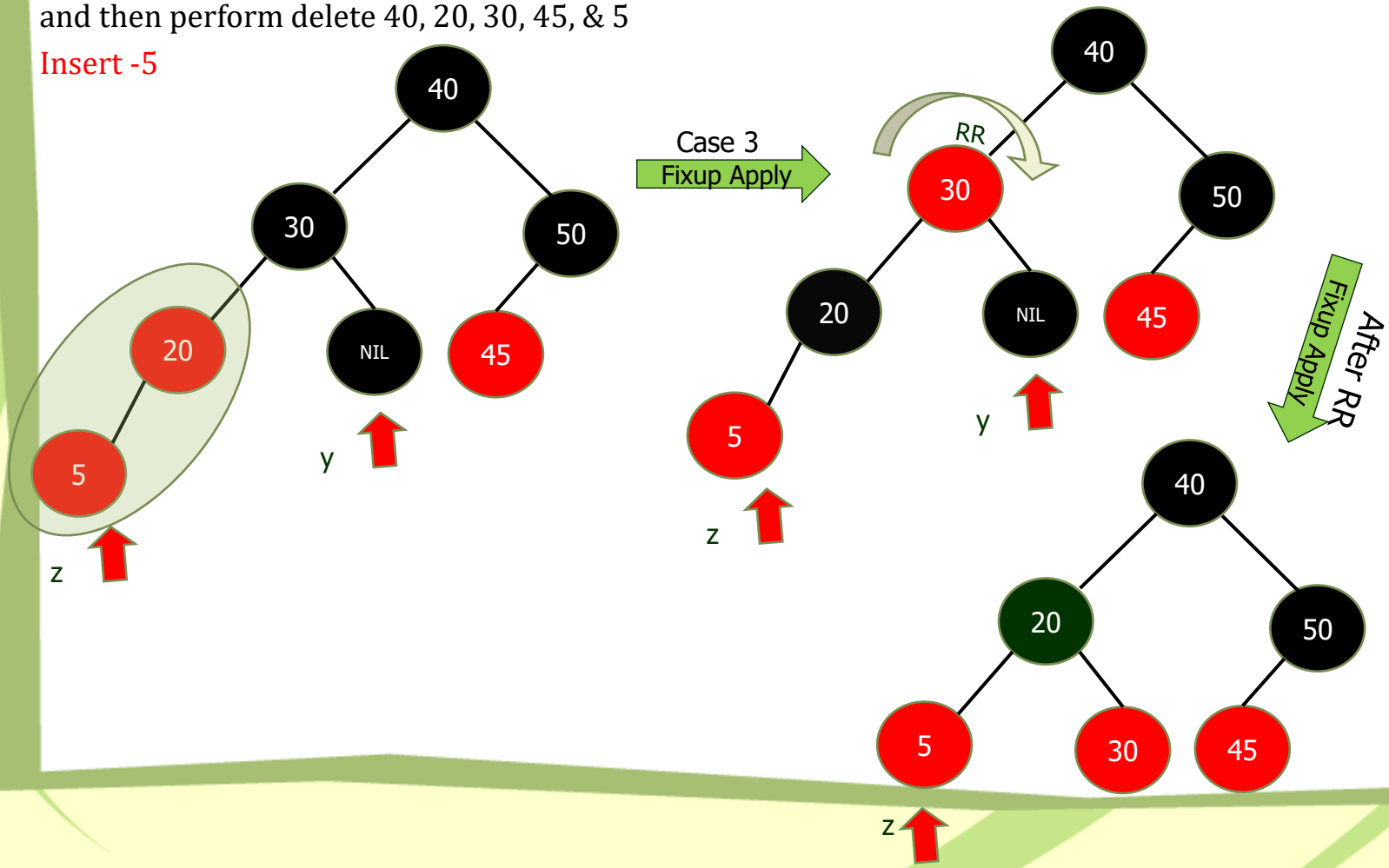
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Insert -5



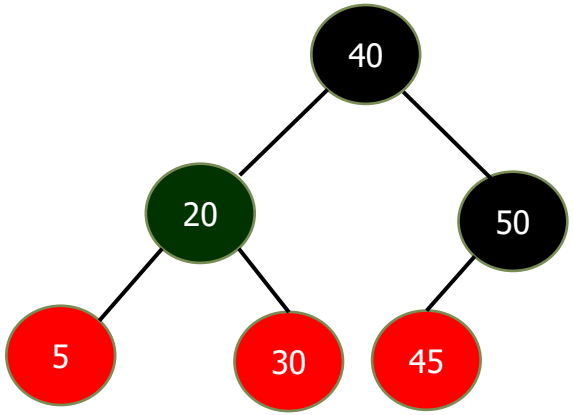
Red Black Tree

Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5



After the Insertion the tree finally looks as shown above.

Now we perform Deletion on RB Tree.

Red Black Tree

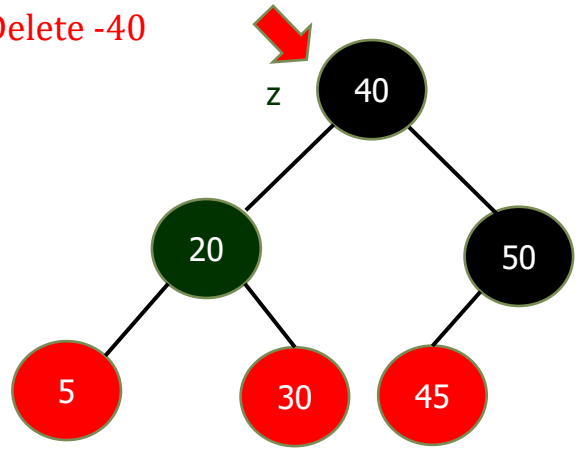
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -40



Red Black Tree

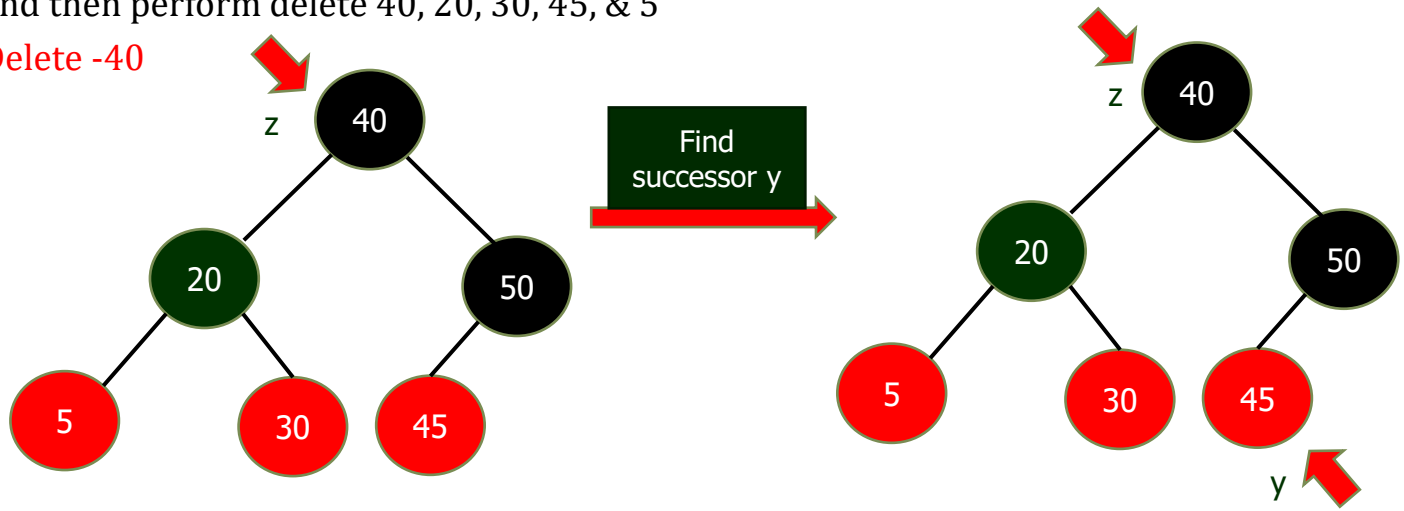
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -40



Red Black Tree

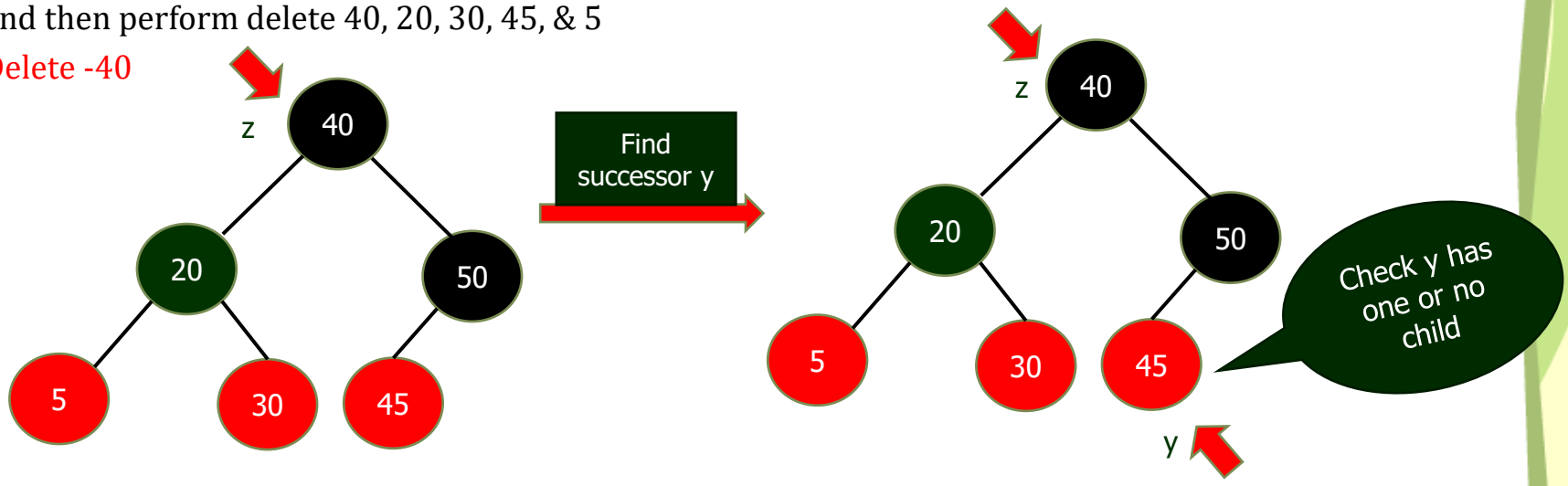
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -40



Red Black Tree

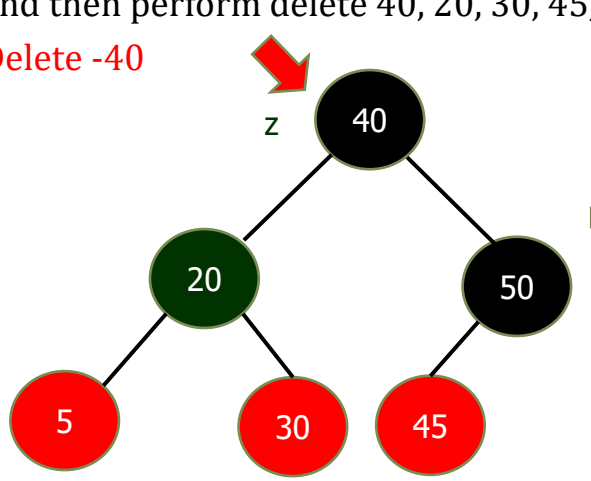
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

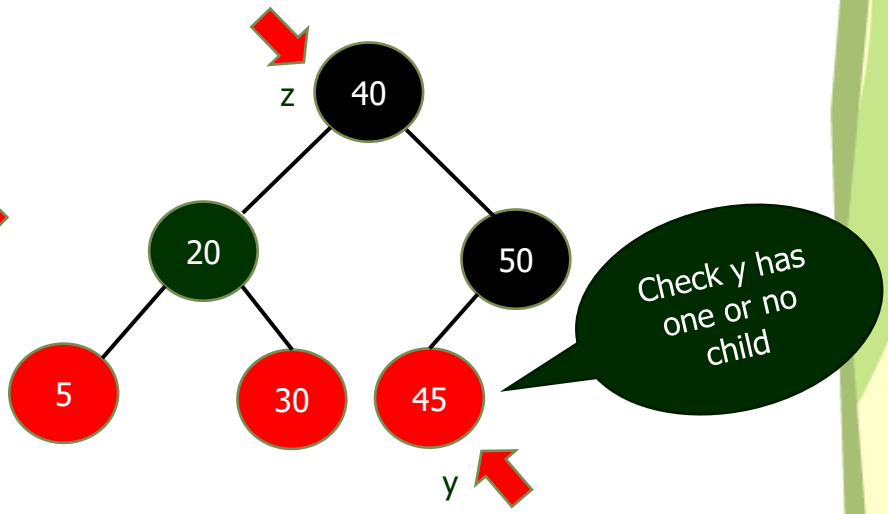
[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -40



Find successor y



If y has no child and then simply exchange the key of y with key of z and delete y

Red Black Tree

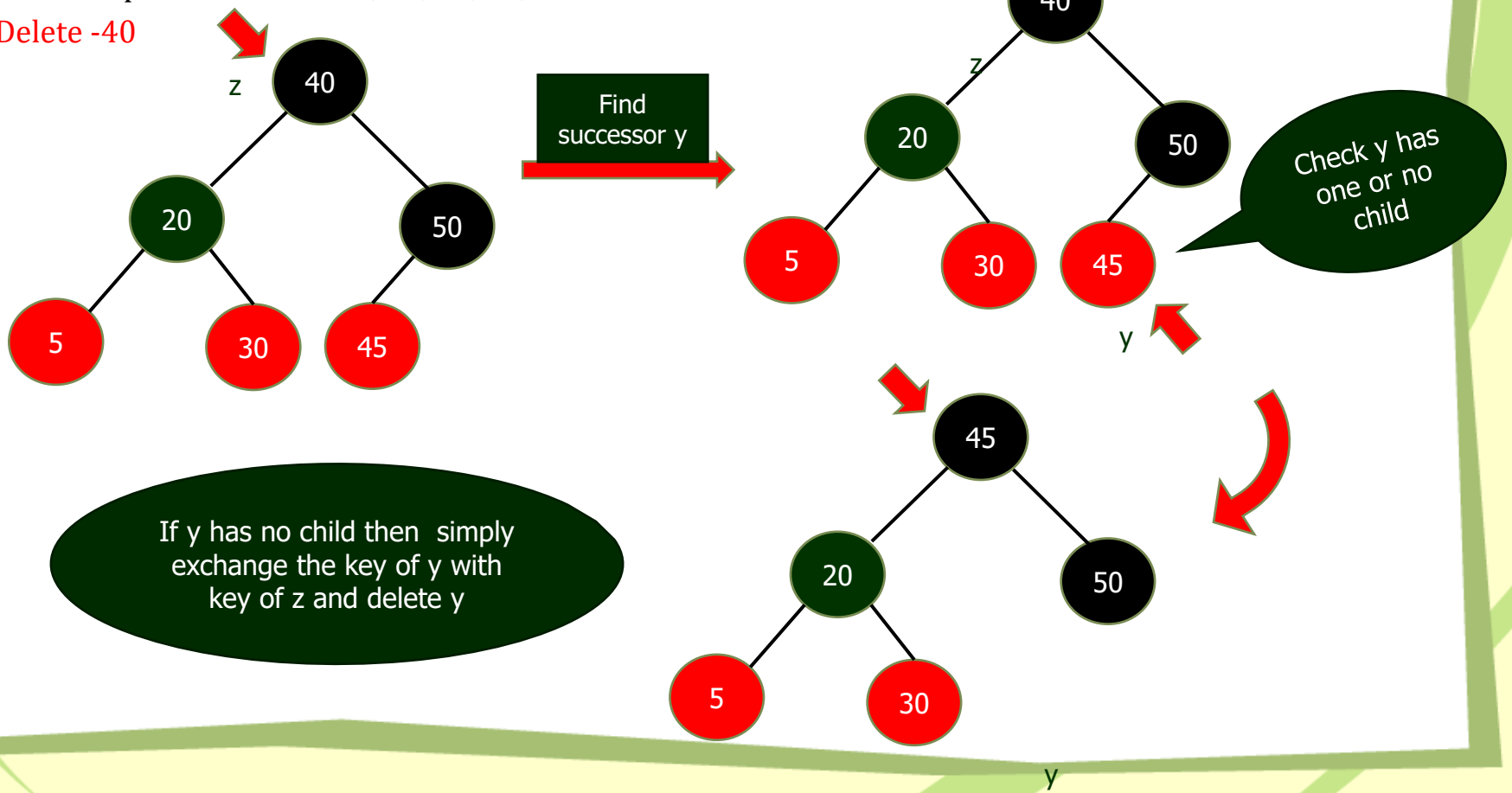
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -40



Red Black Tree

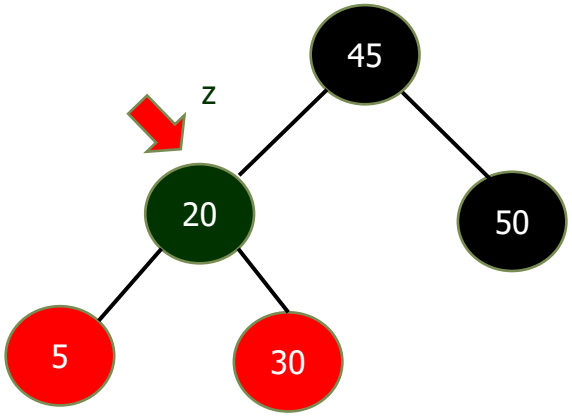
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -20



Red Black Tree

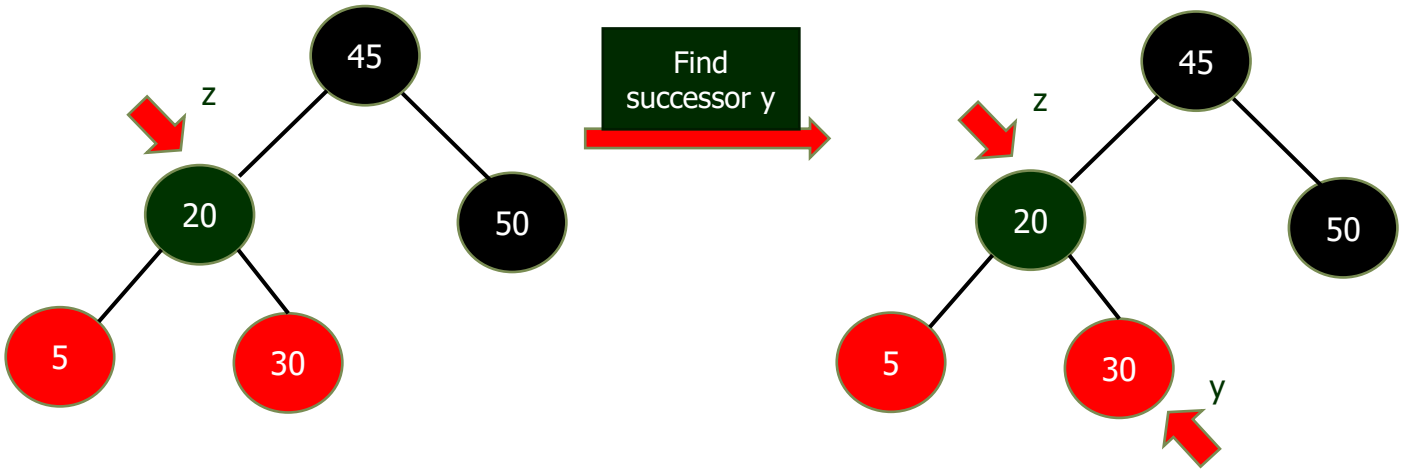
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -20



Red Black Tree

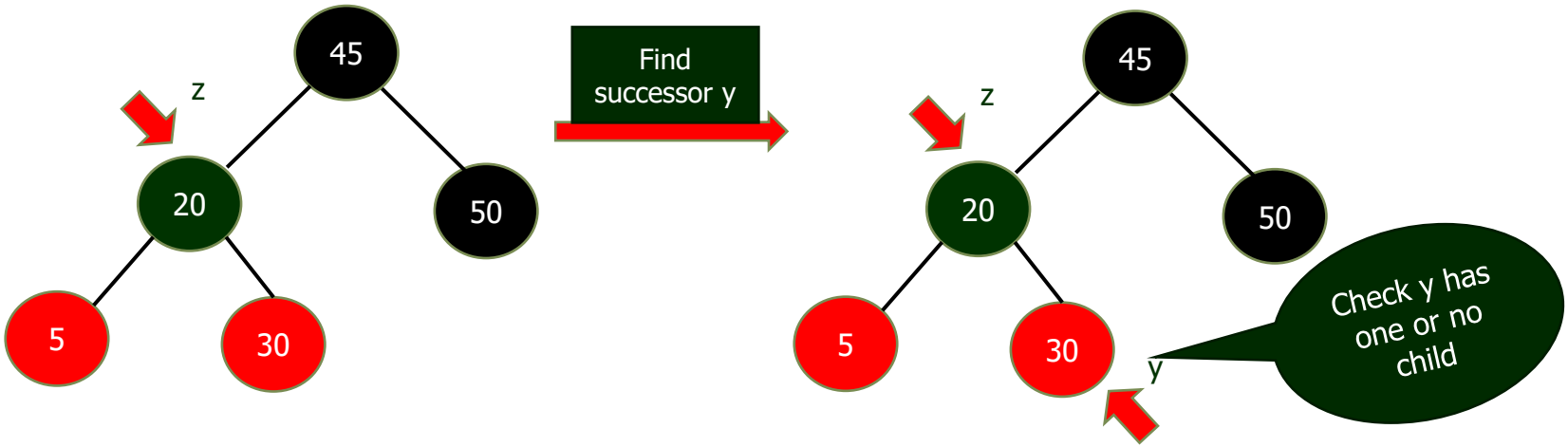
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -20



Red Black Tree

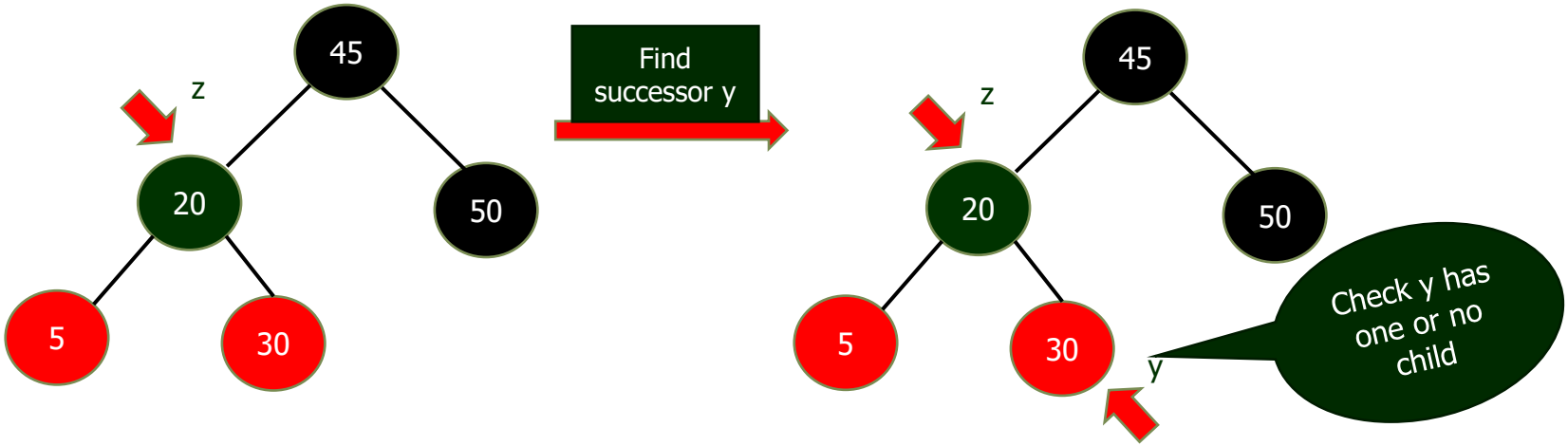
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -20



If y has no child and then simply exchange the key of y with key of z and delete y

Red Black Tree

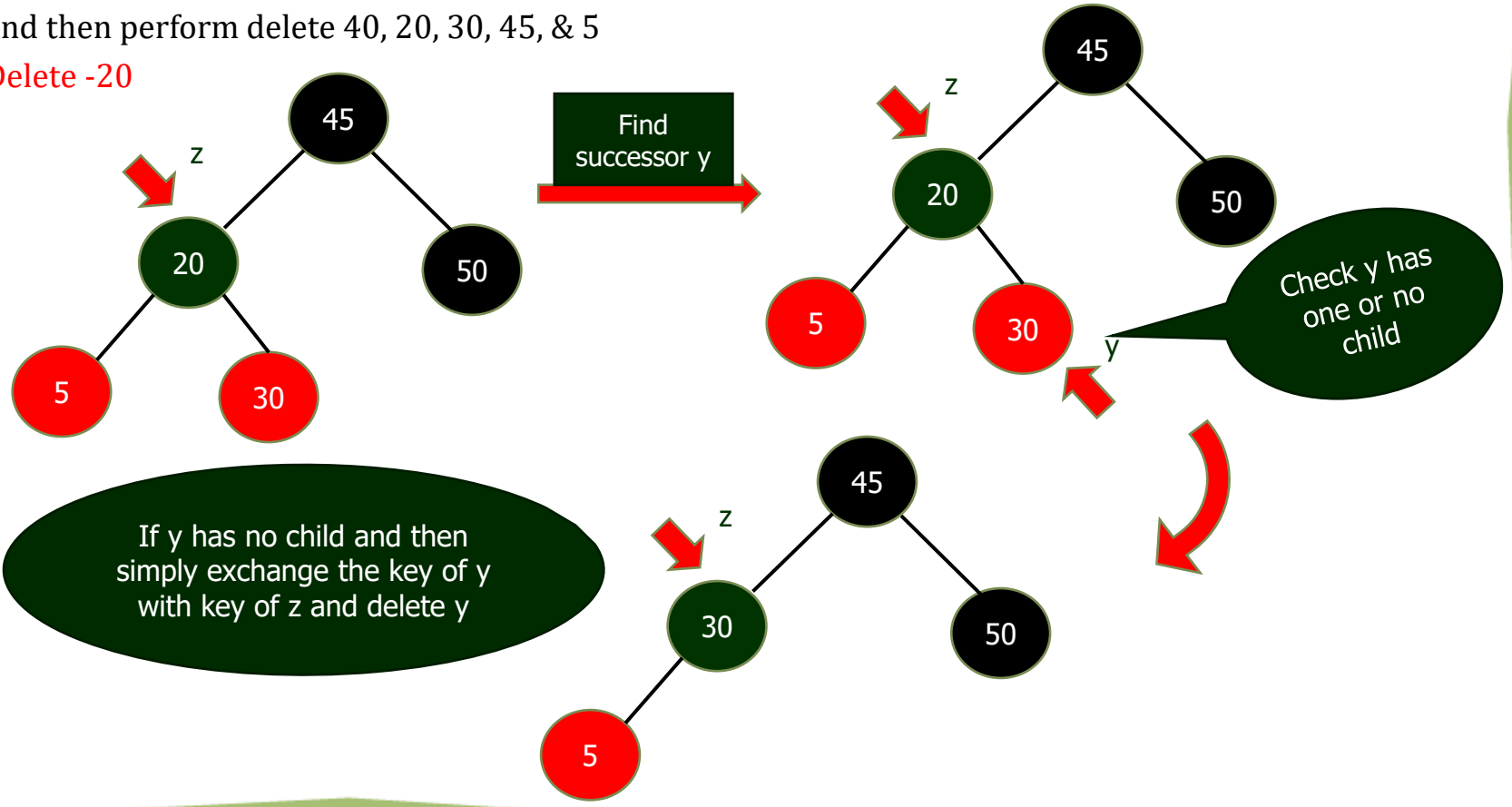
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -20



Red Black Tree

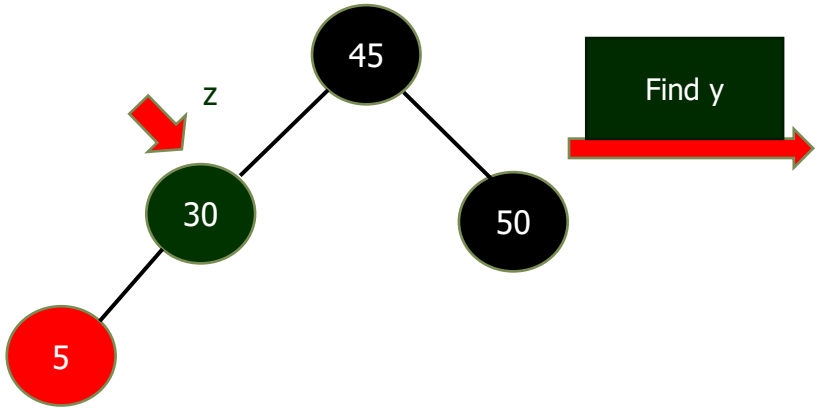
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -30



Red Black Tree

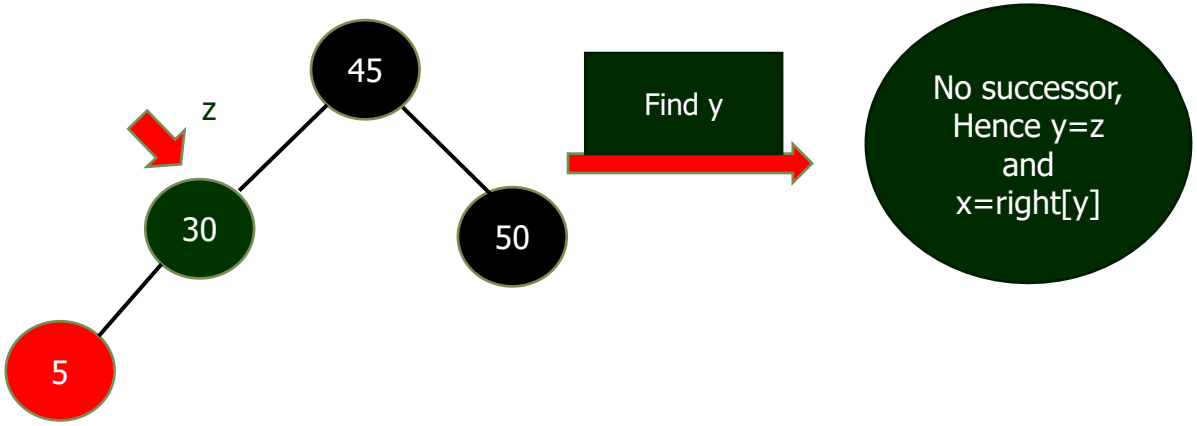
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -30



Red Black Tree

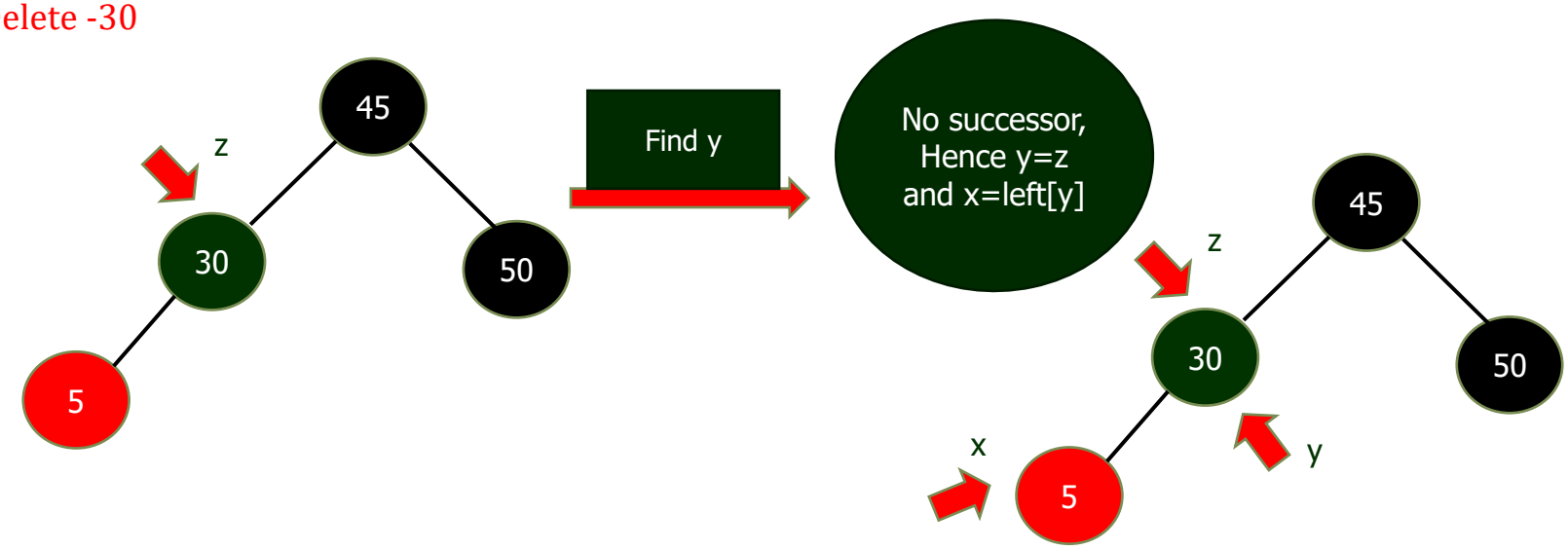
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -30



Red Black Tree

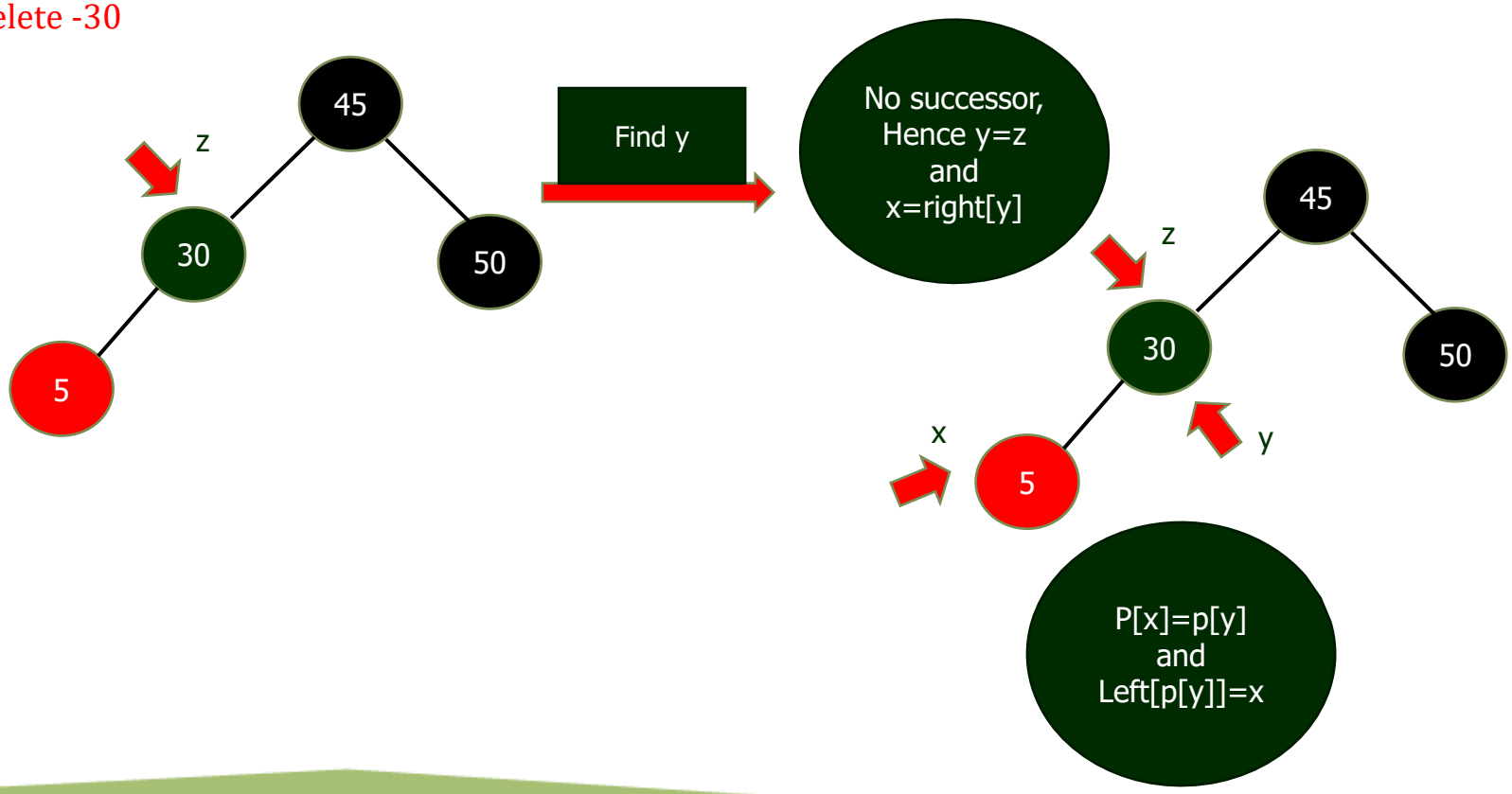
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -30



Red Black Tree

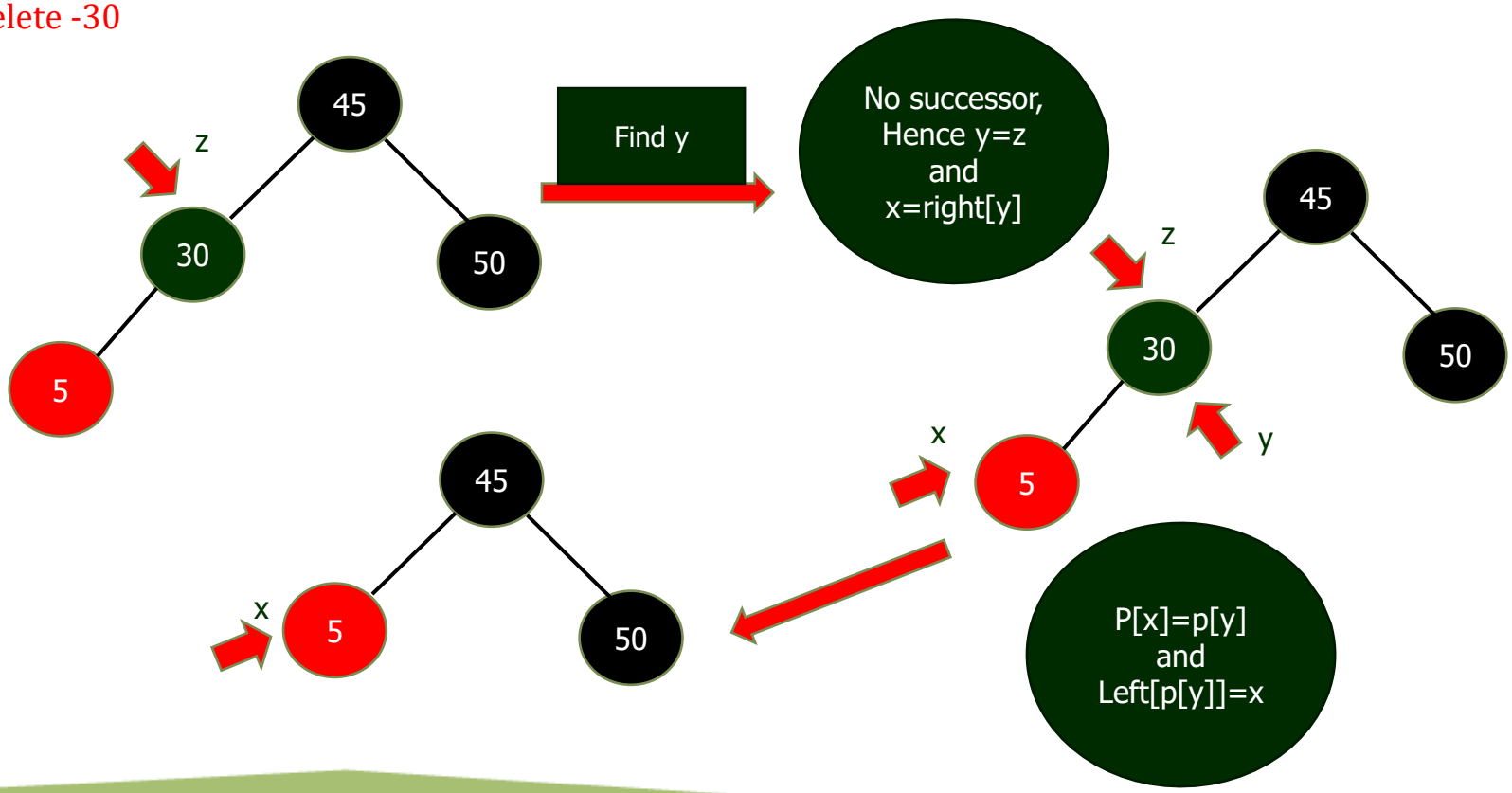
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -30



Red Black Tree

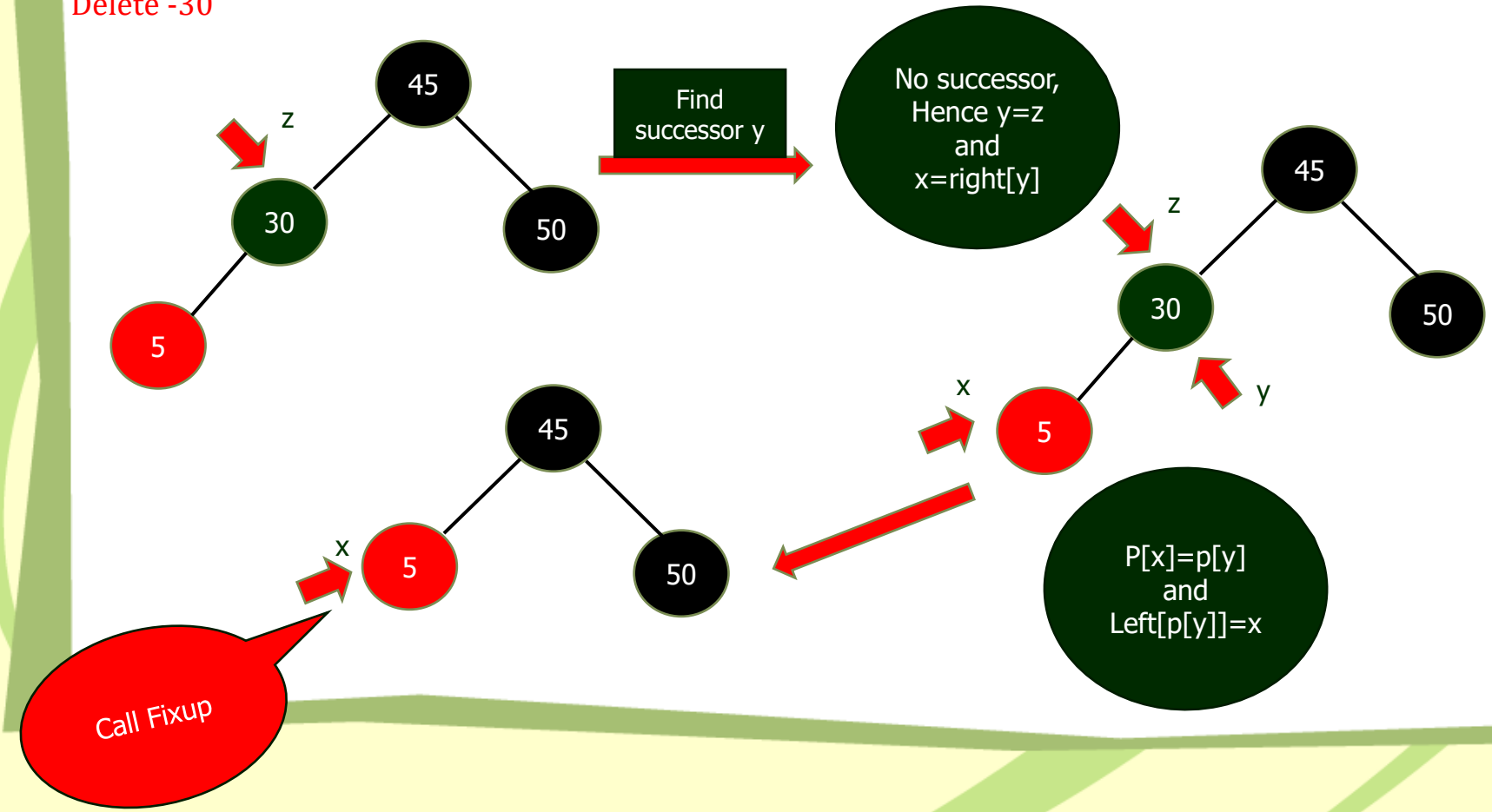
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -30



Red Black Tree

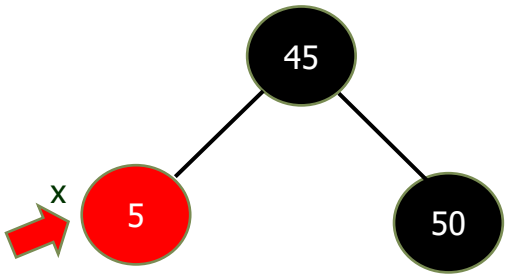
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -30



It was found that it was not enter inside the while loop, so execute the last line of RB-Delete-Fixup,(i.e. color[x]=Black

Red Black Tree

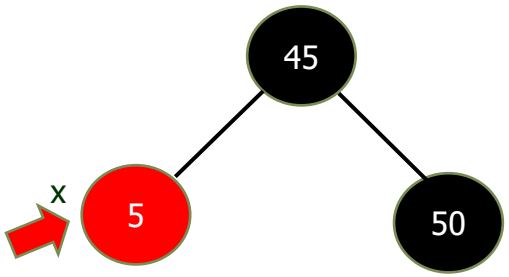
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

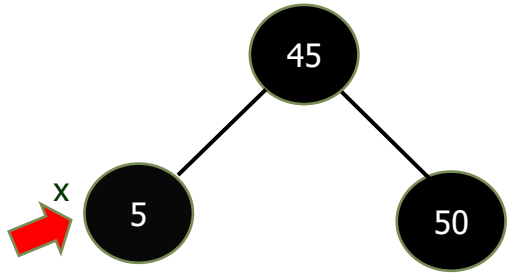
[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -30



It was found that it was not enter inside the while loop, so execute the last line of RB-Delete-Fixup,(i.e. color[x]=Black



Red Black Tree

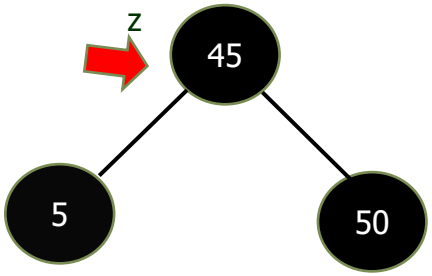
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

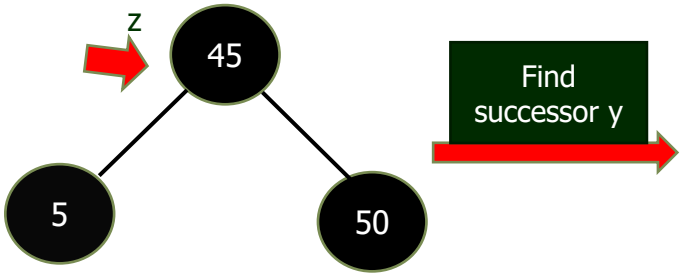
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

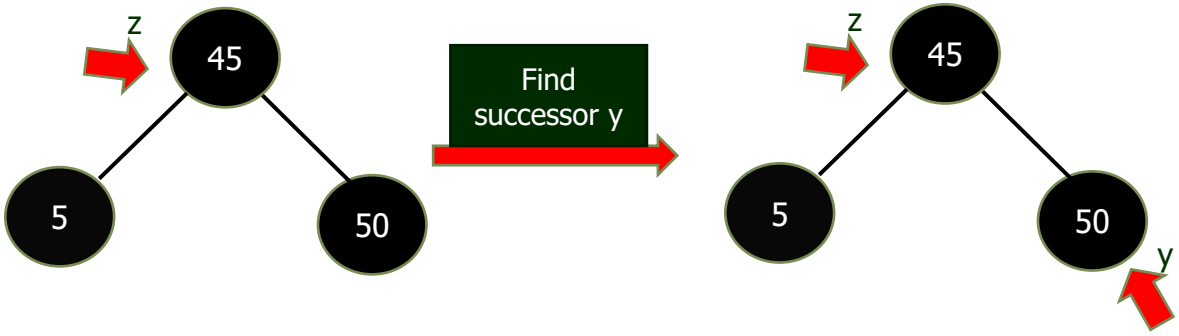
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

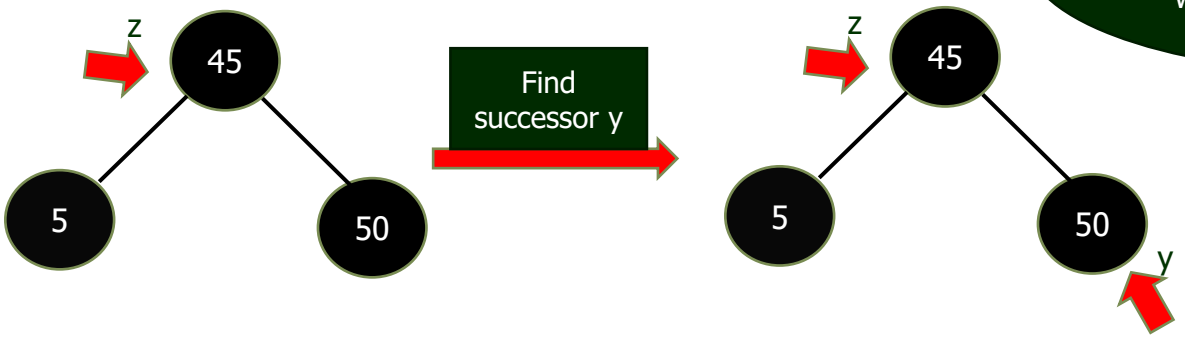
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -40



If y has no child and then simply exchange the key of y with key of z and delete y

Red Black Tree

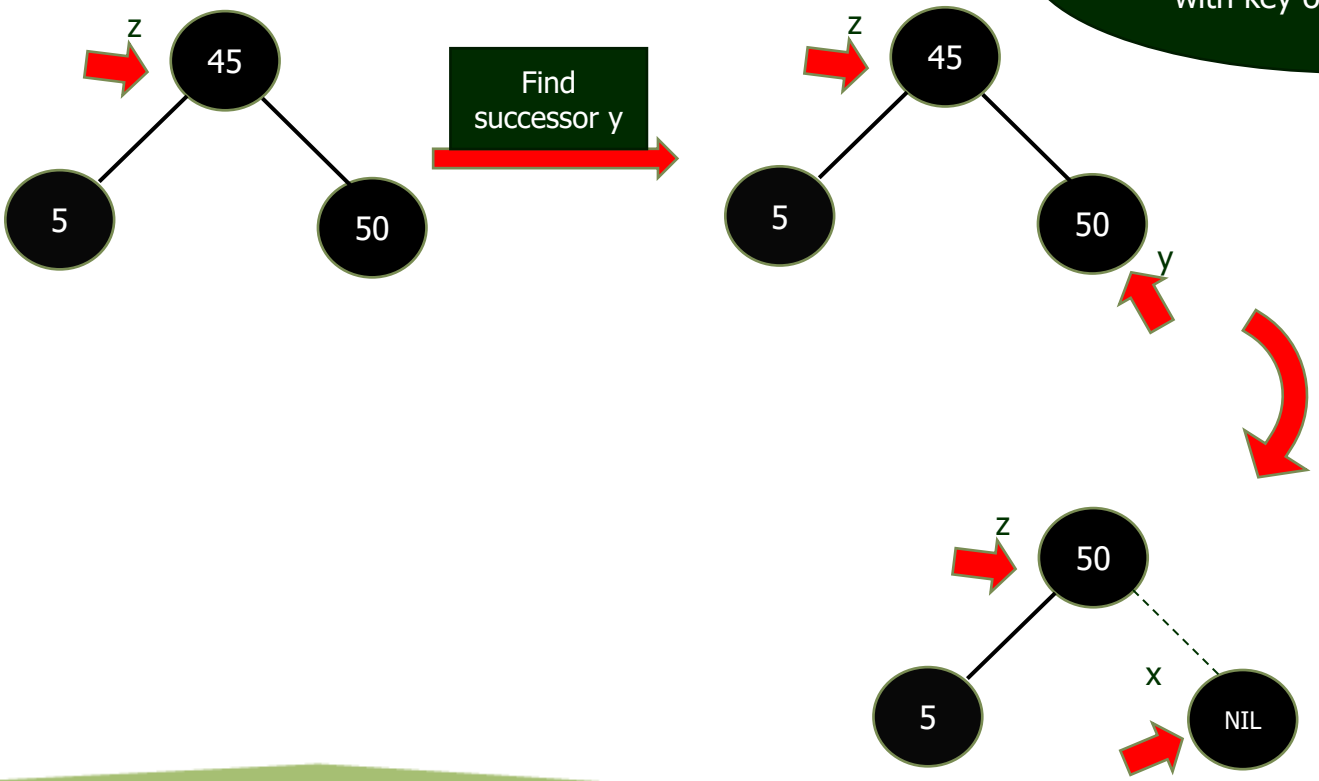
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

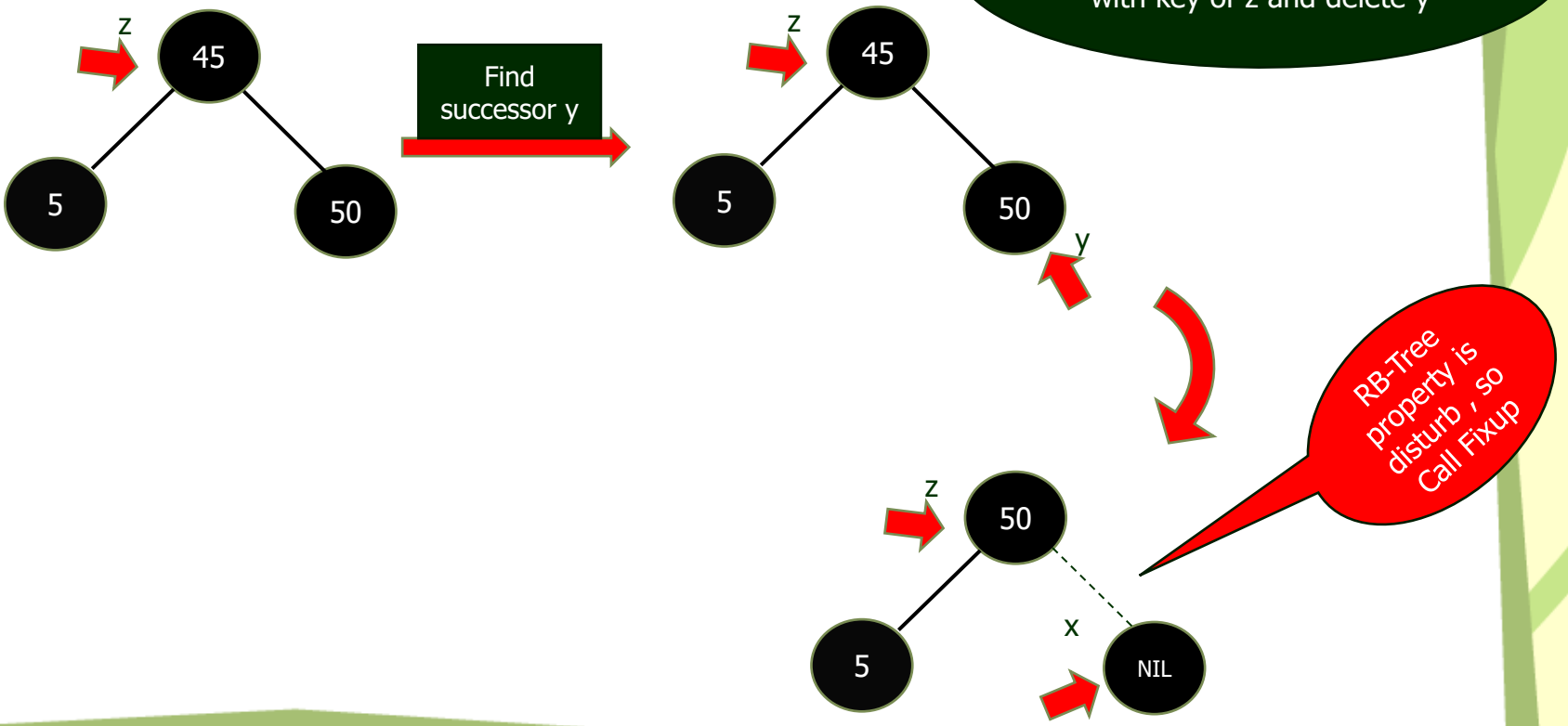
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

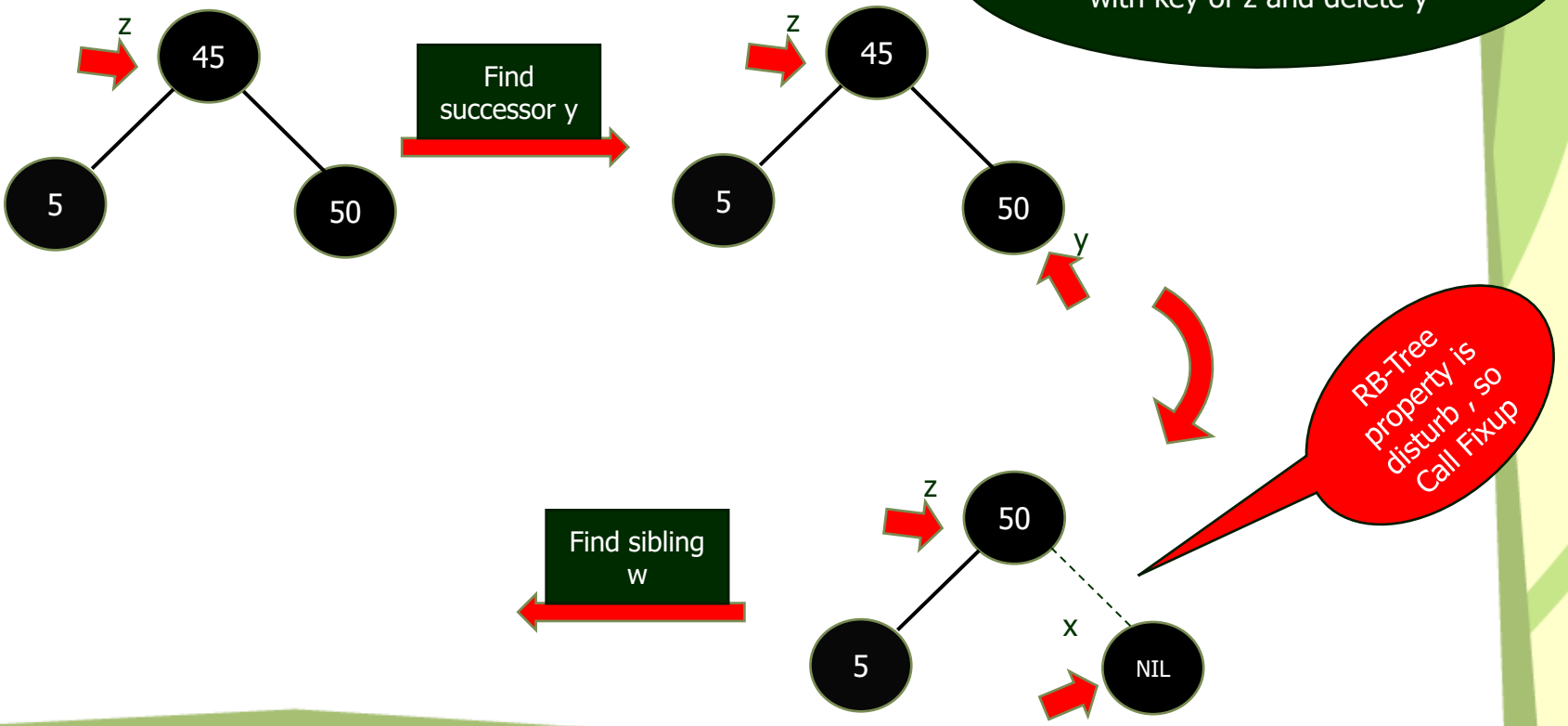
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

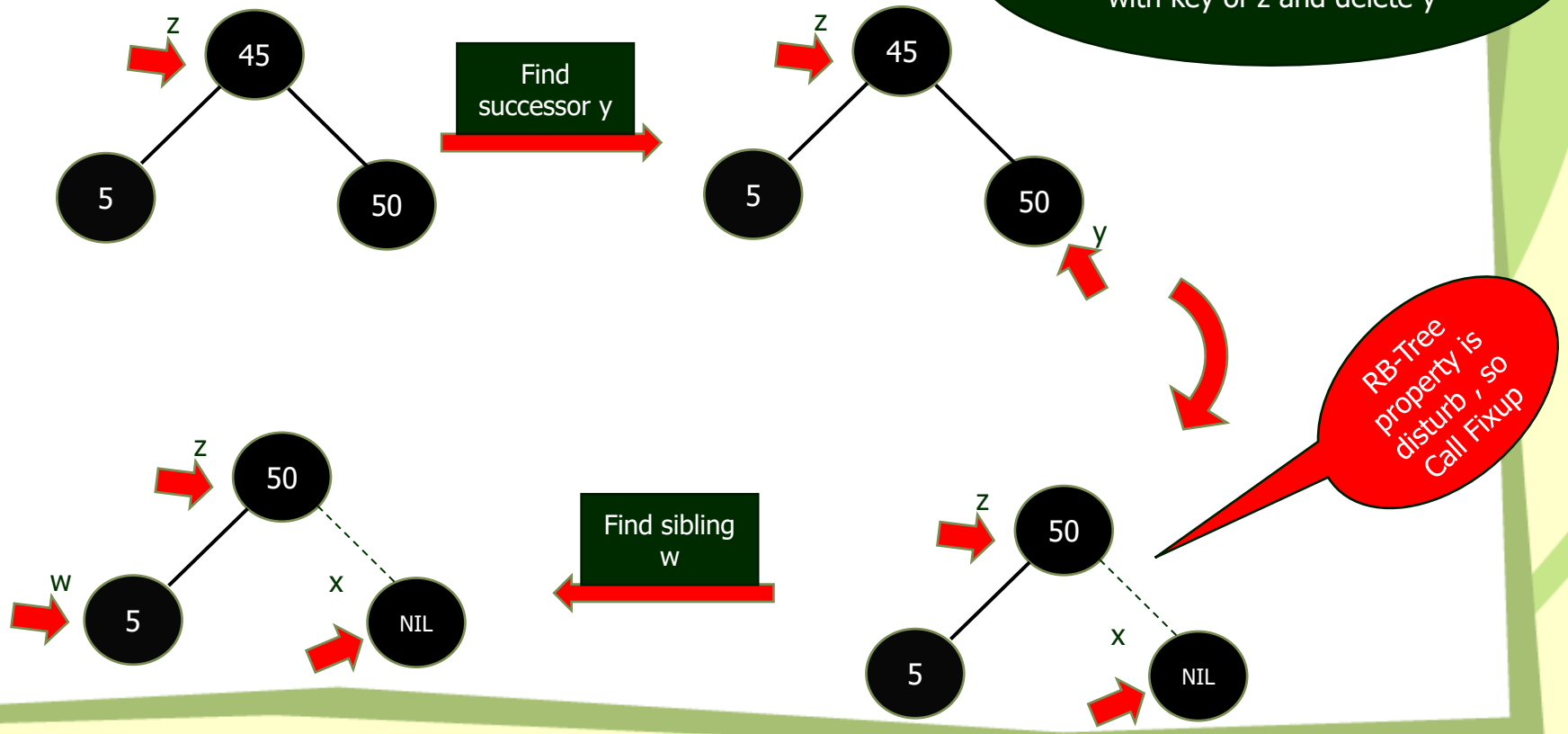
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

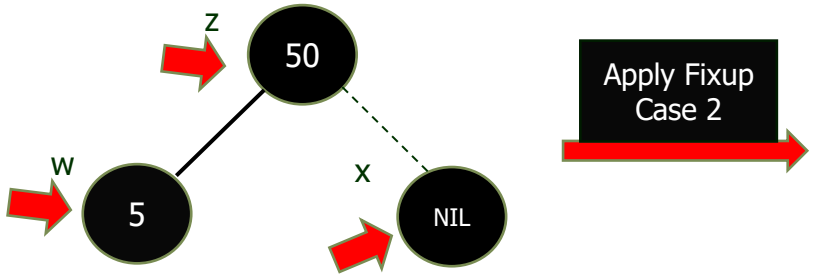
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

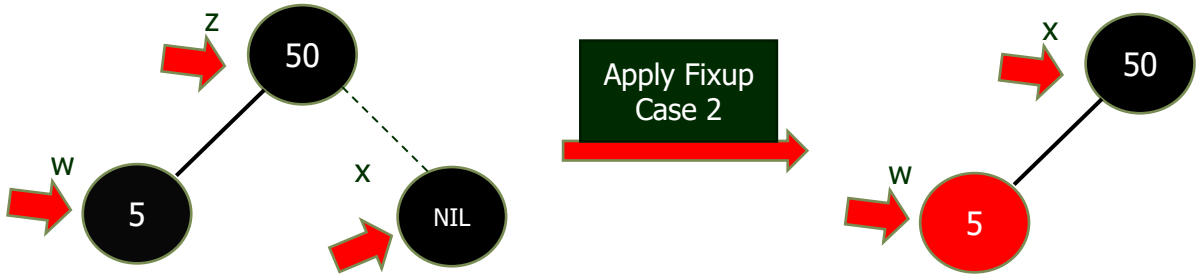
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -45



Red Black Tree

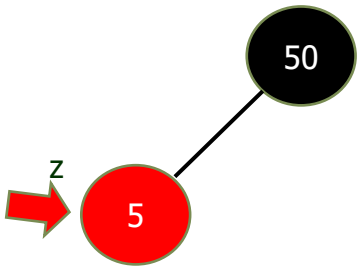
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -5



Red Black Tree

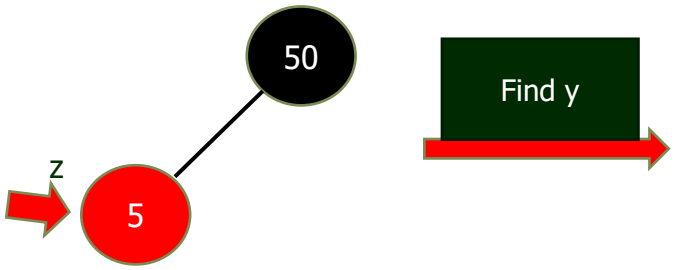
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -5



Red Black Tree

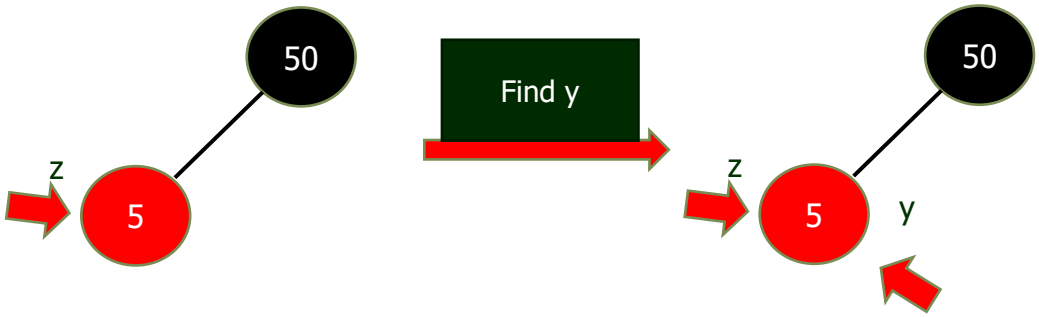
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -5



Red Black Tree

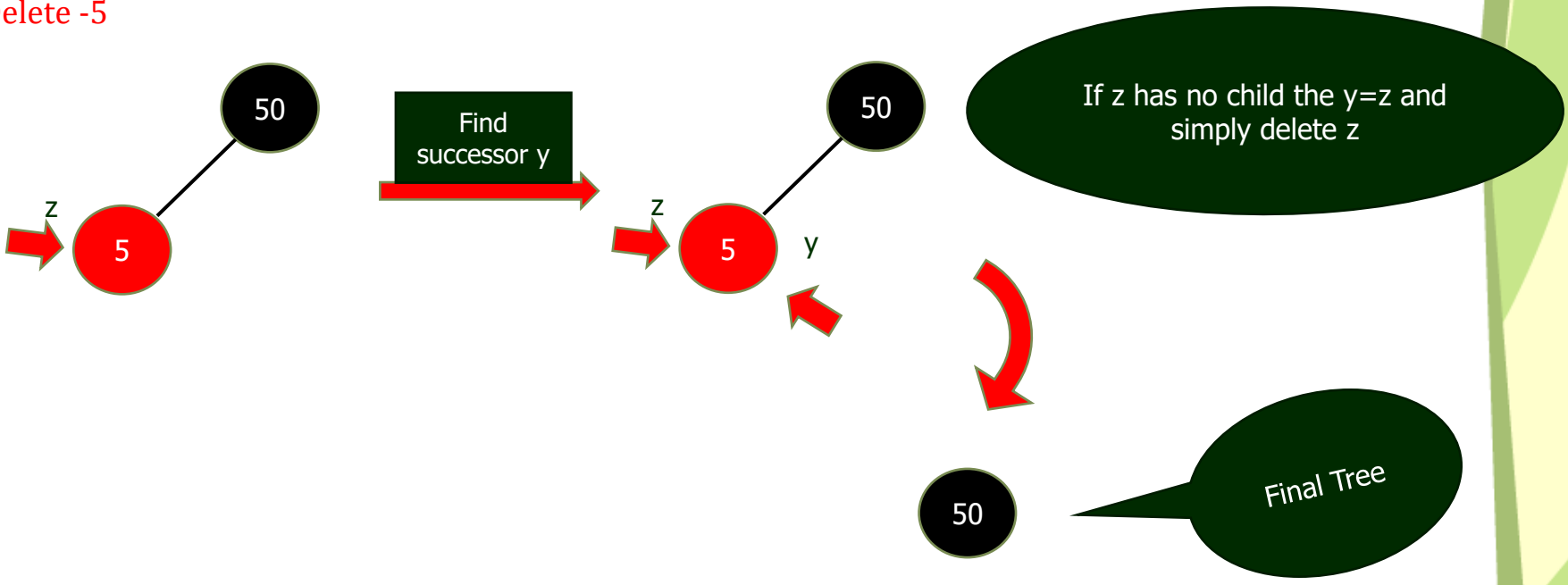
Insertion (Example 1):

Insert the following elements into an empty RB-Tree.

[50, 40, 30, 45, 20, 5]

and then perform delete 40, 20, 30, 45, & 5

Delete -5

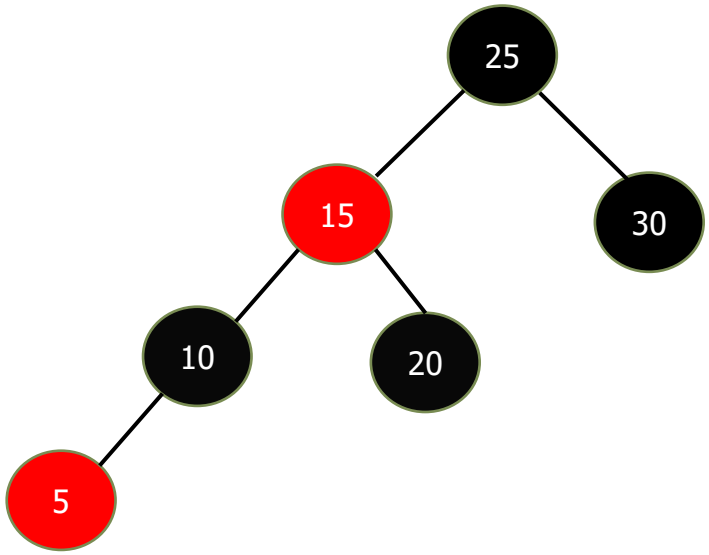


Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>



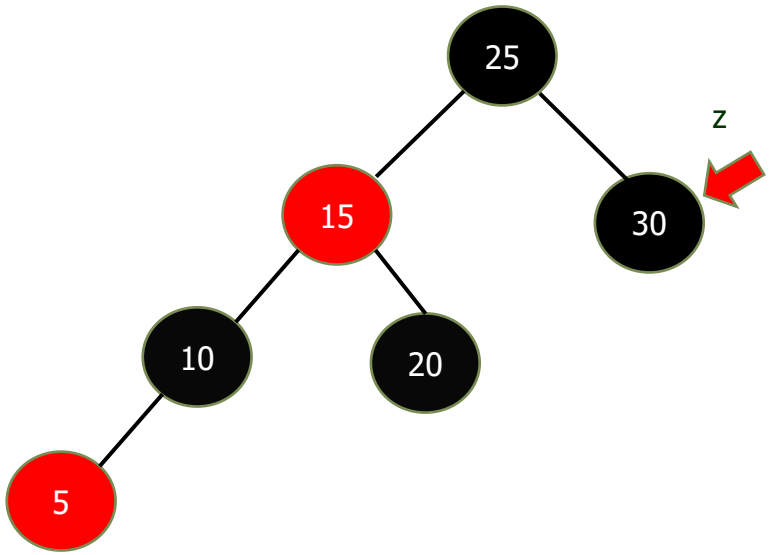
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -30



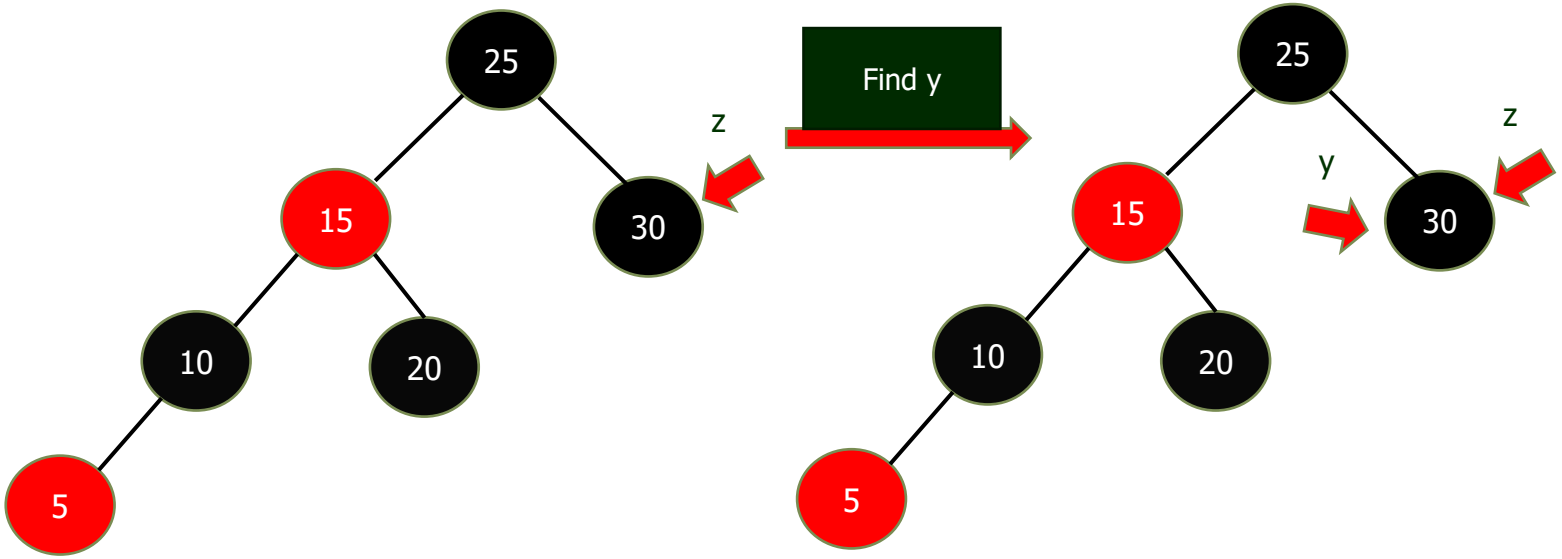
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -30



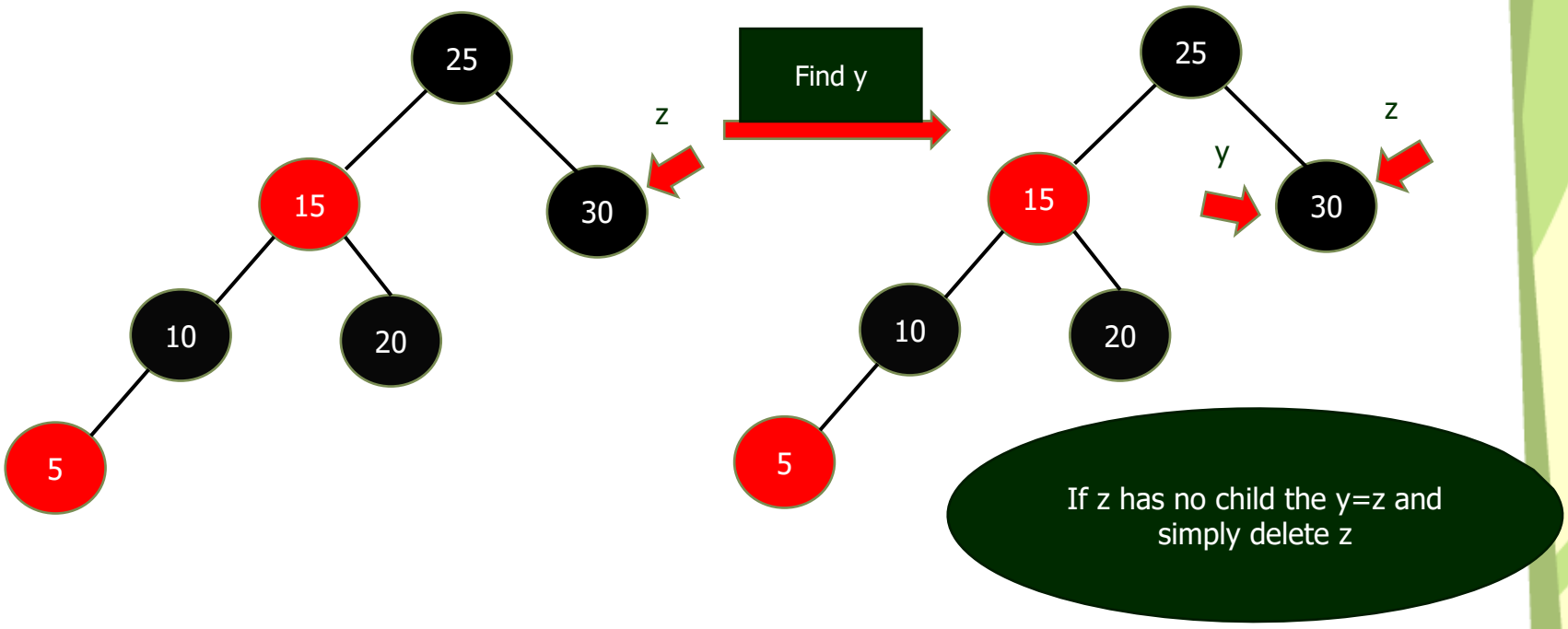
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -30



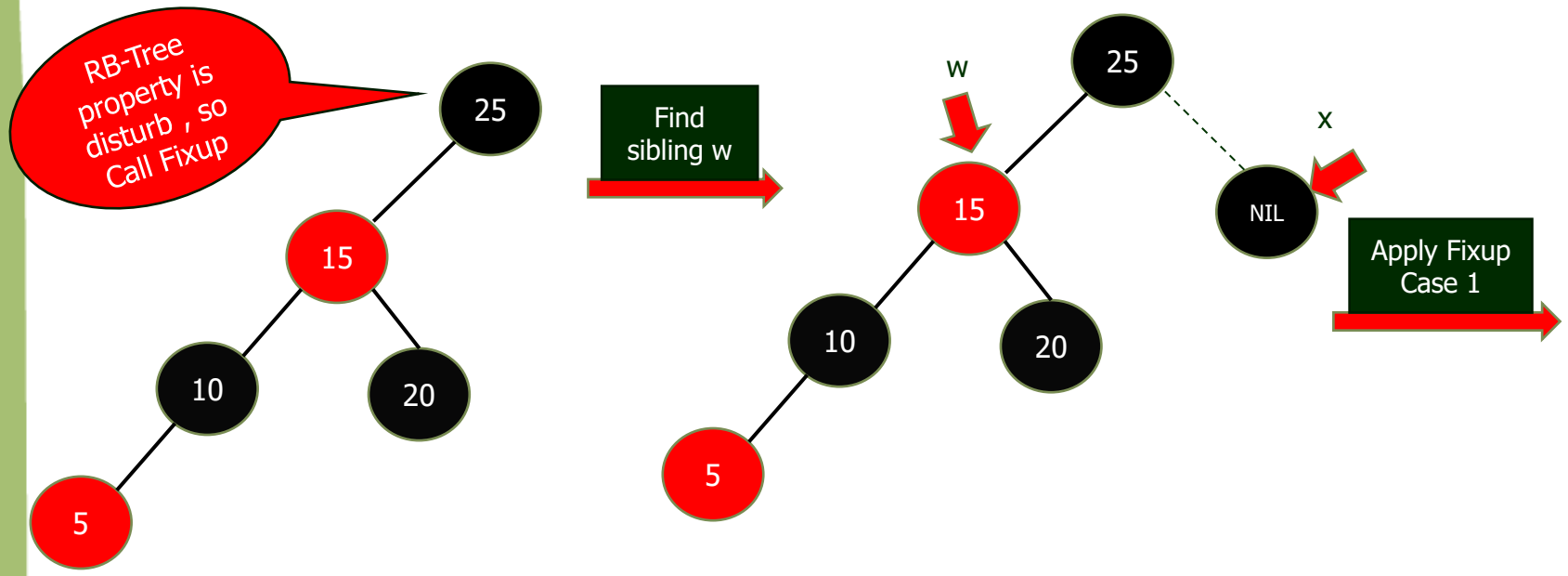
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -30



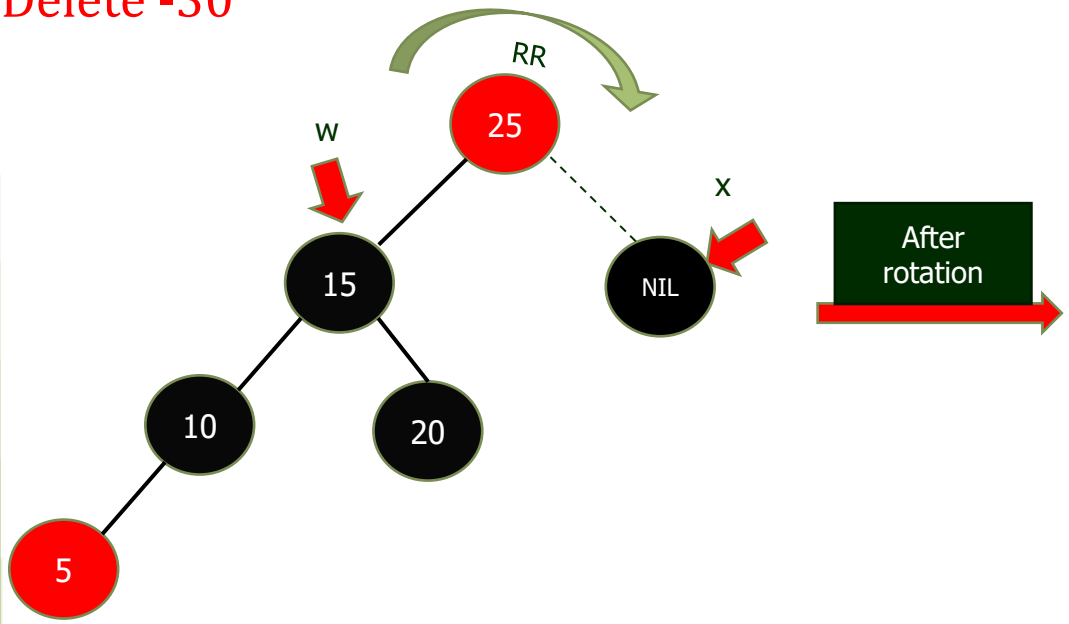
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -30



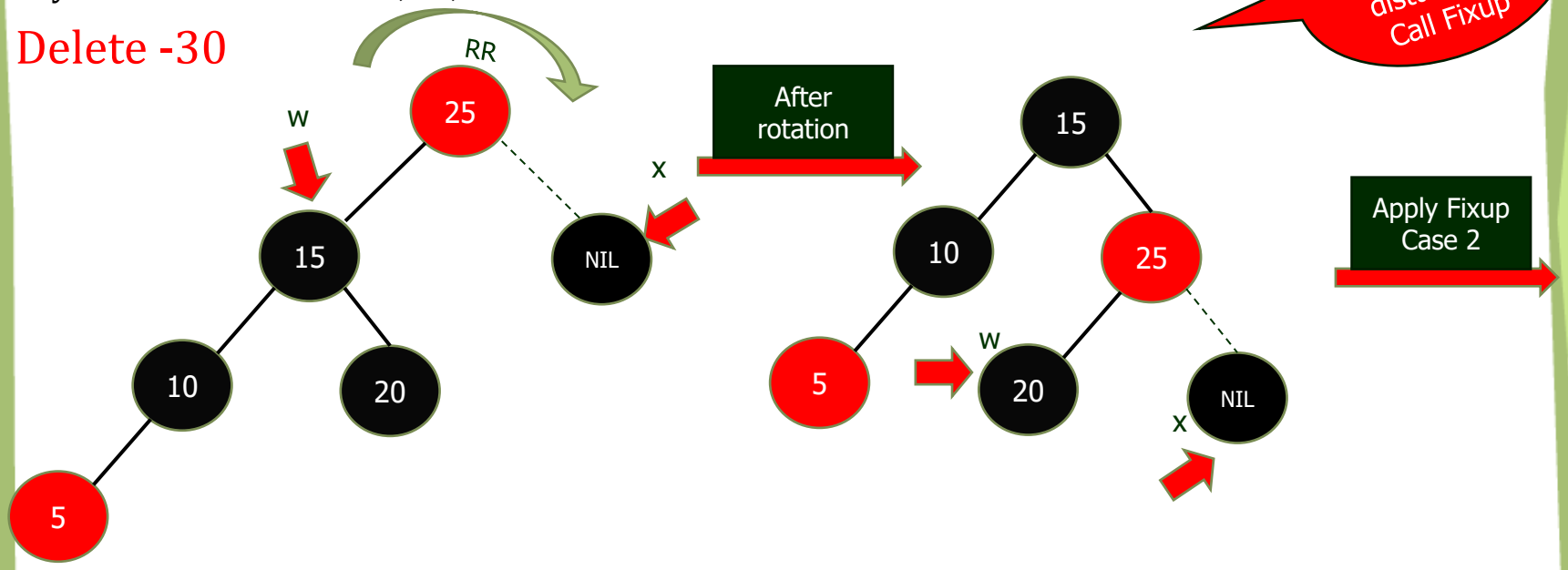
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -30



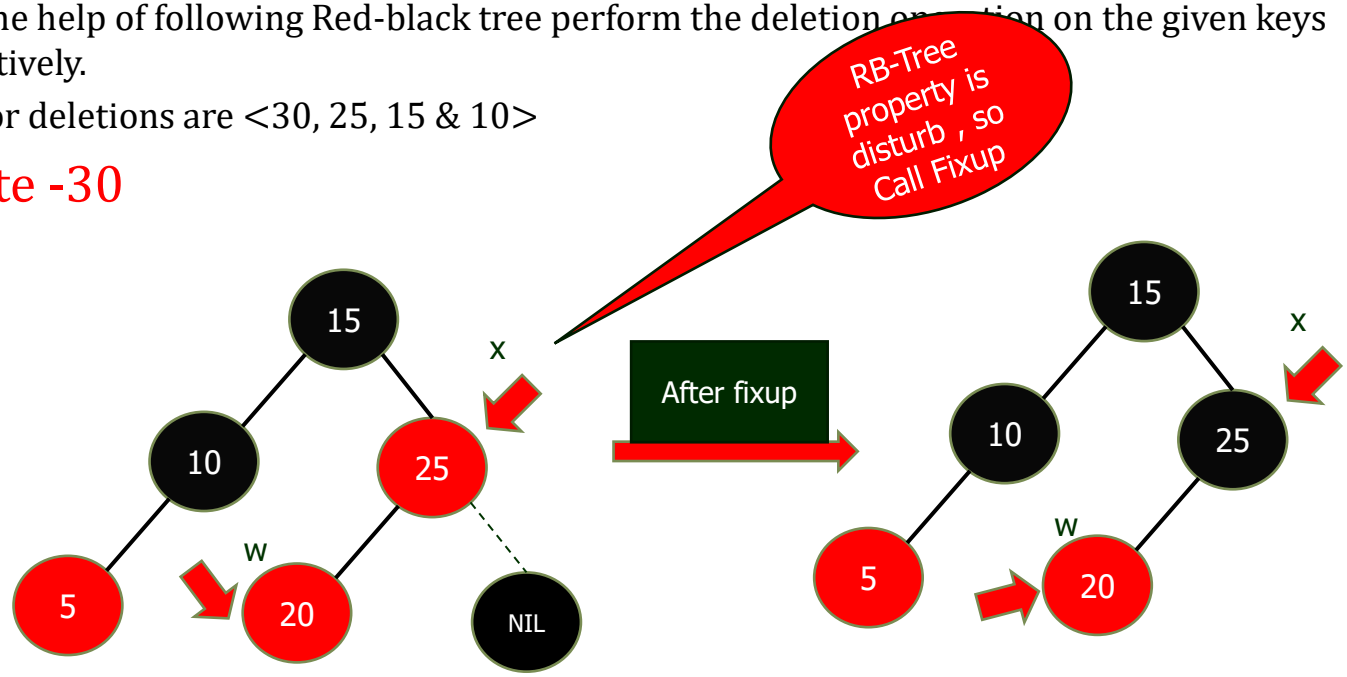
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -30



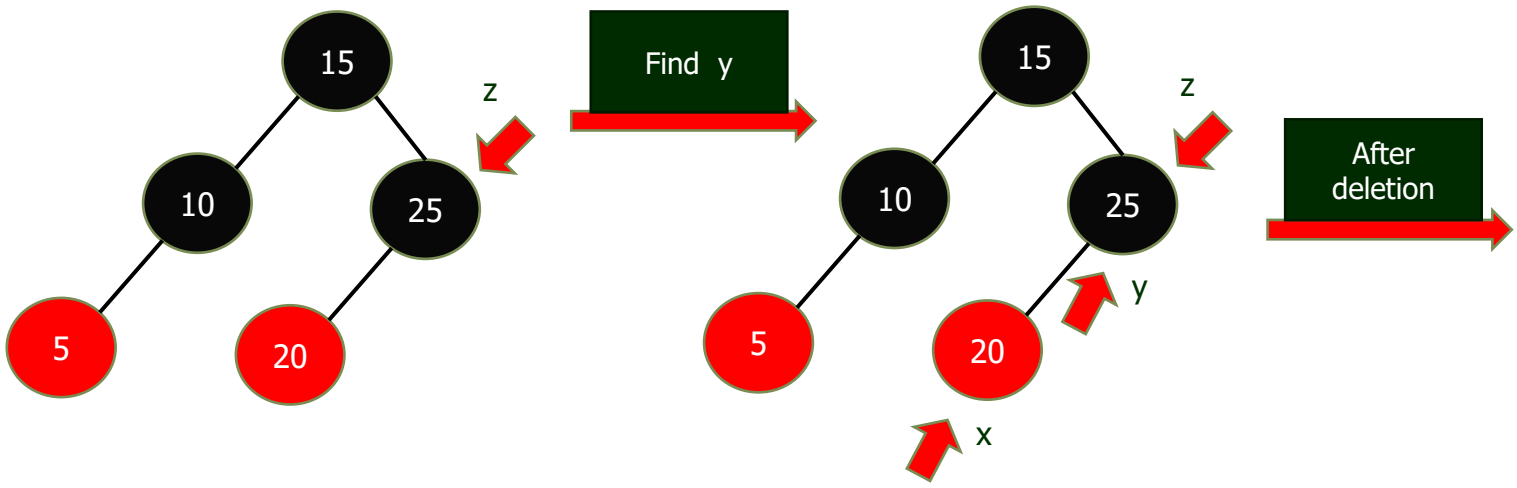
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -25



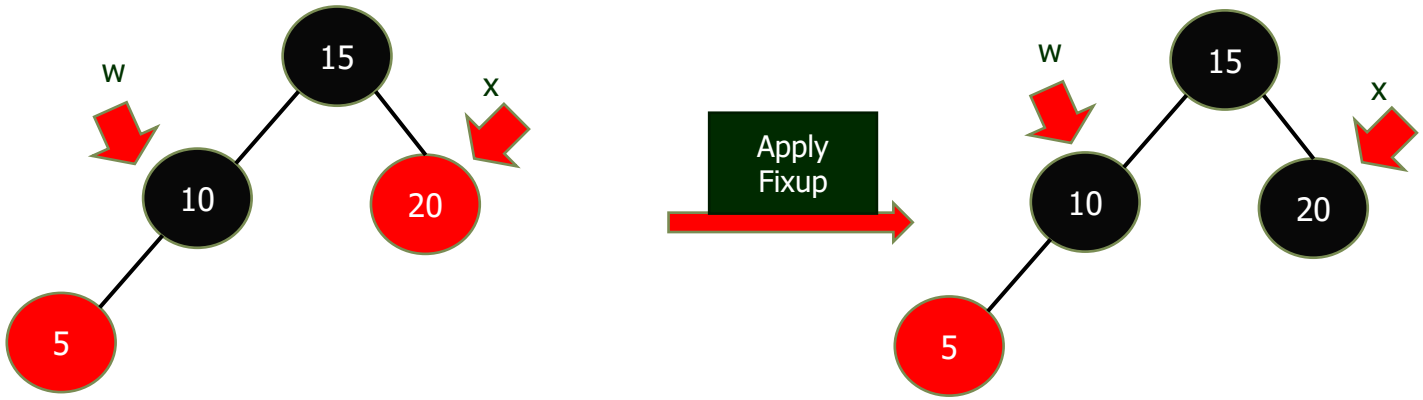
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -25



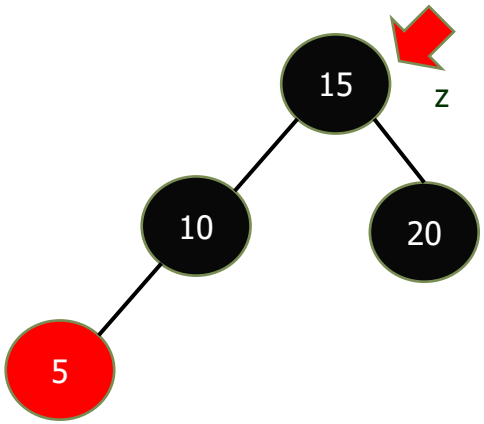
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -15



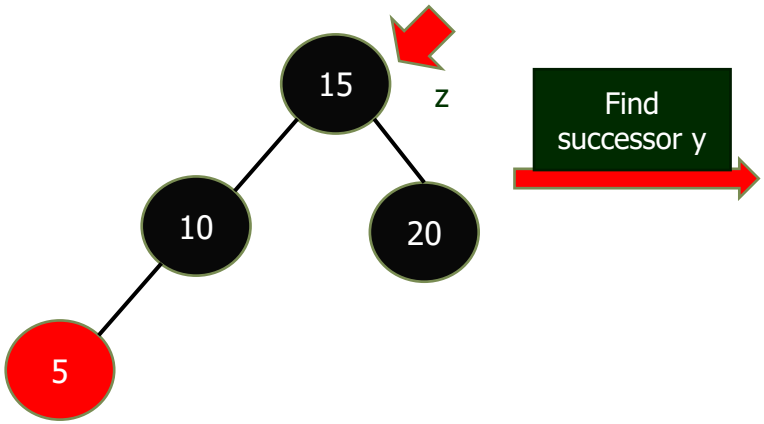
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -15



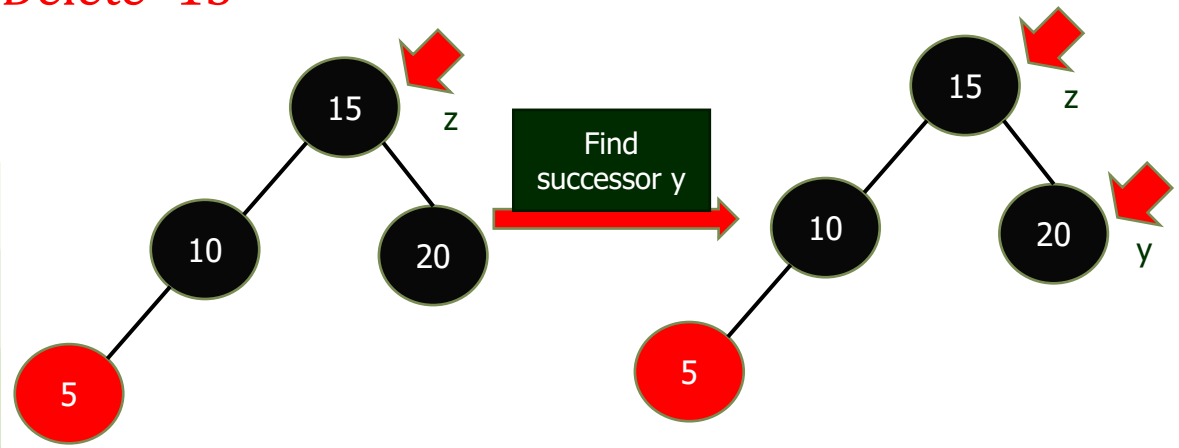
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -15



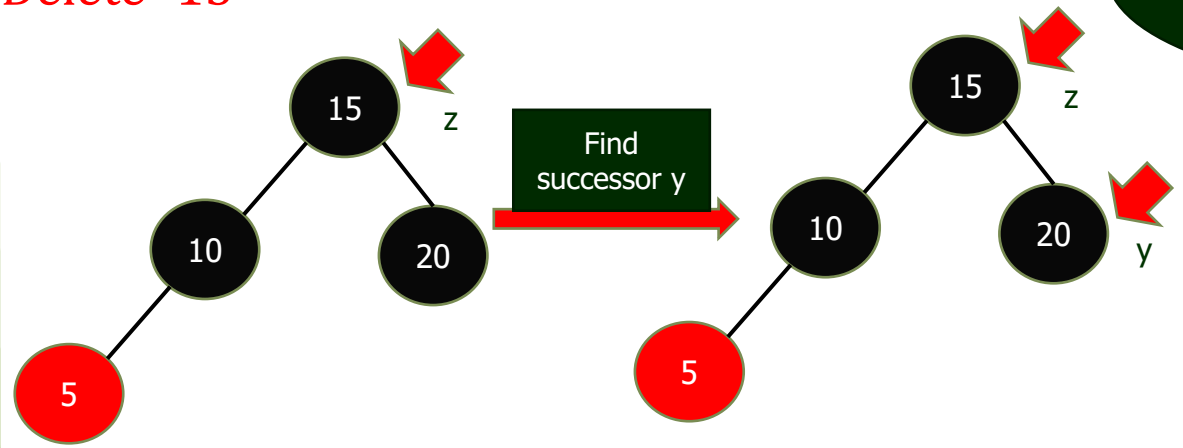
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -15



If y has no child and then simply exchange the key of y with key of z and delete y

Red Black Tree

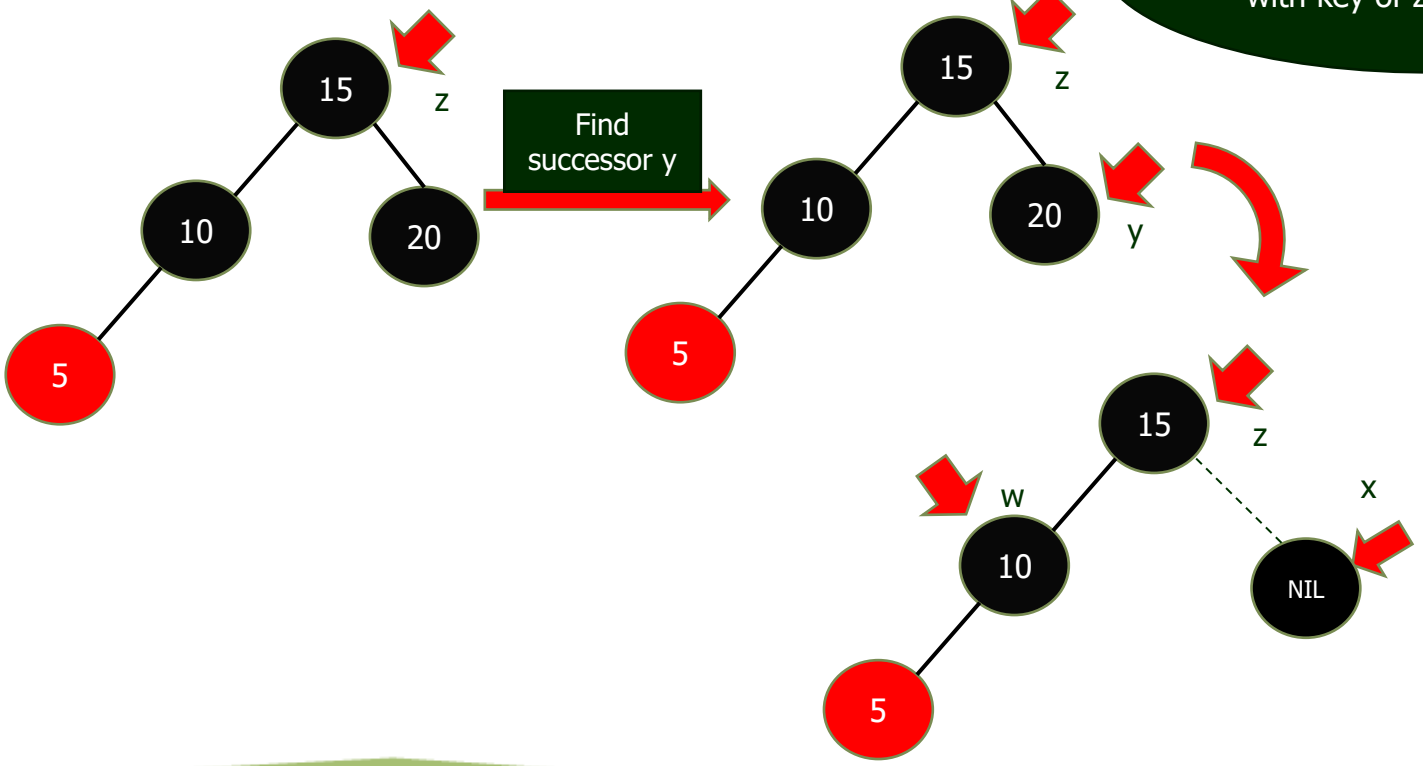
Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -15

If y has no child and then simply exchange the key of y with key of z and delete y



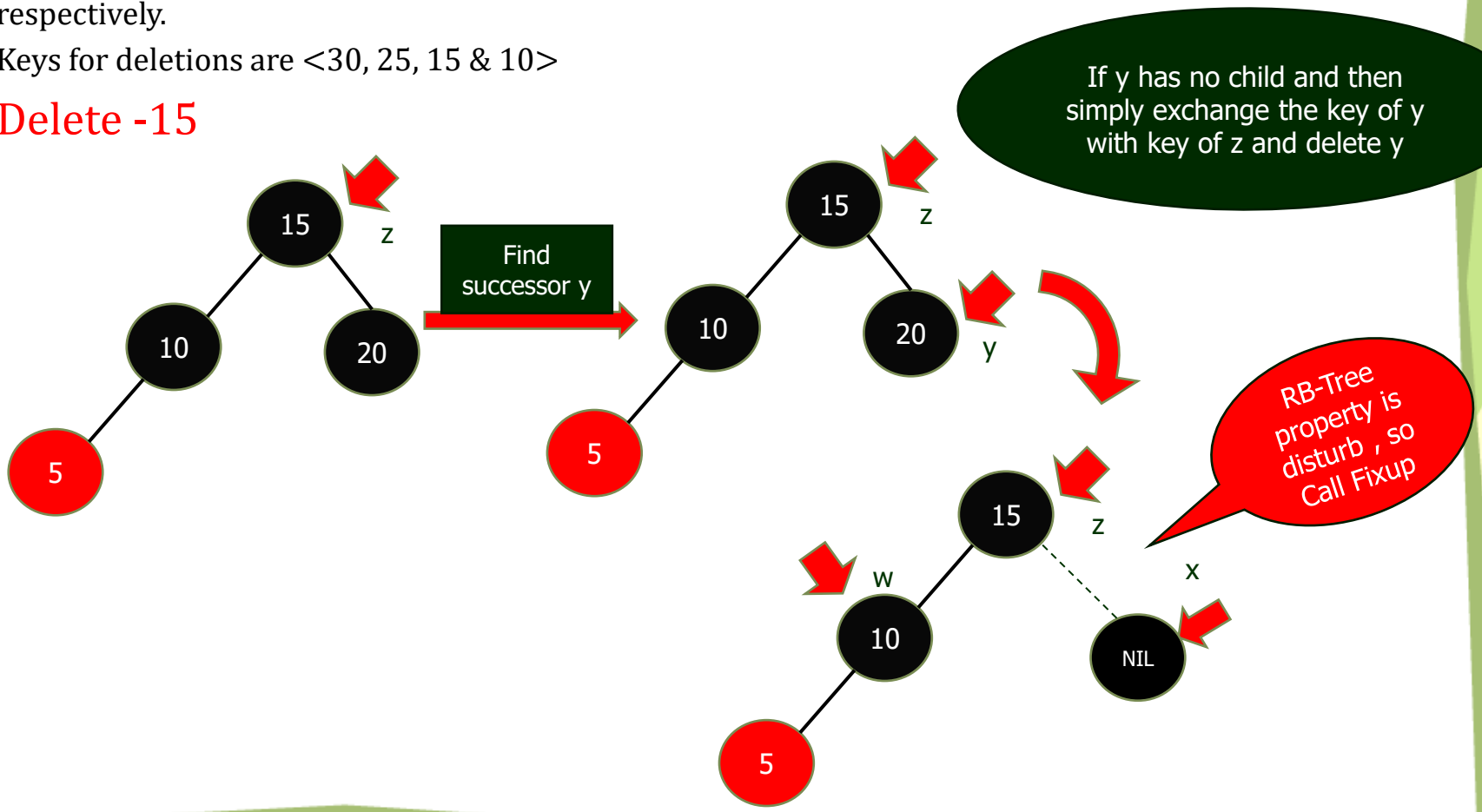
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -15



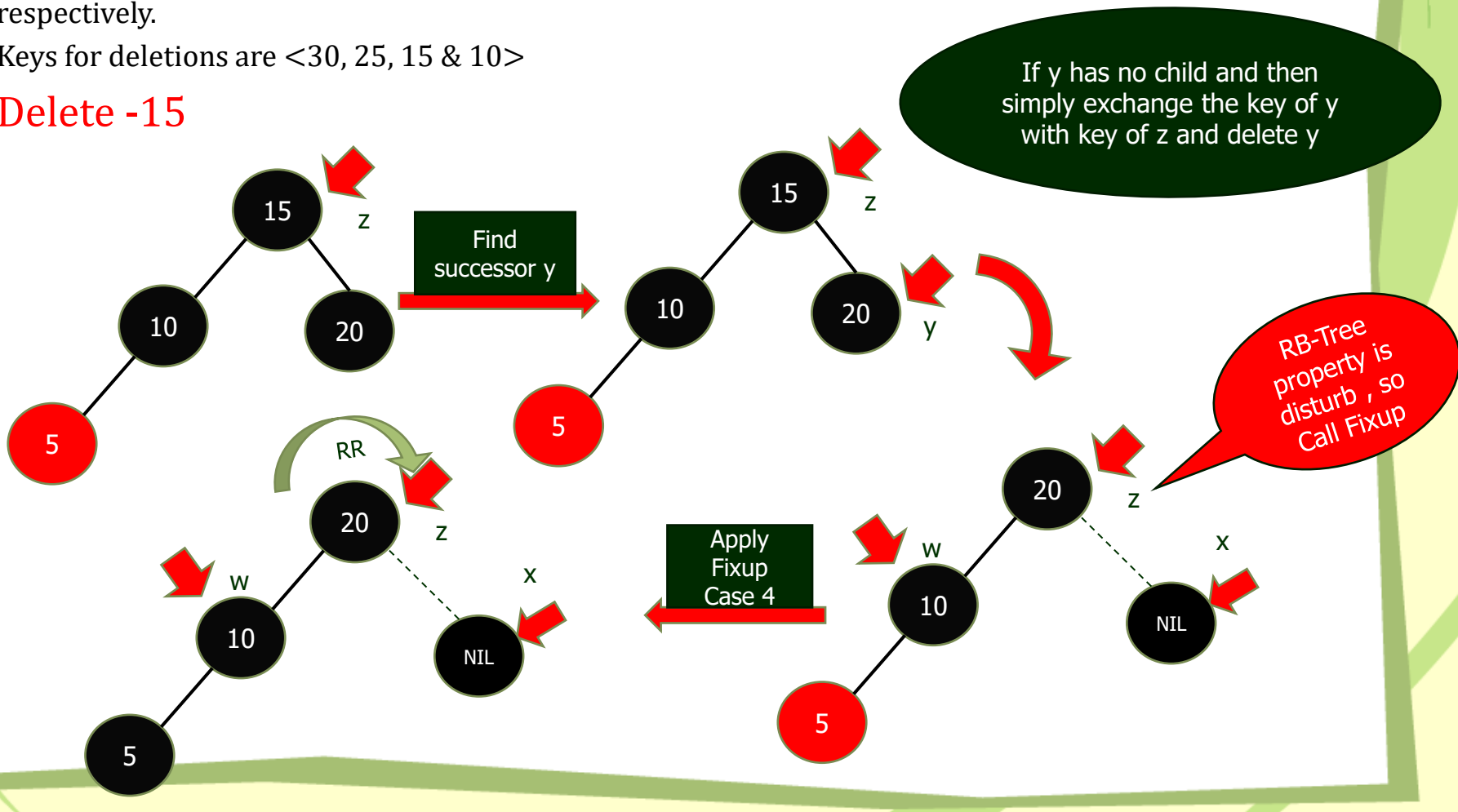
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -15



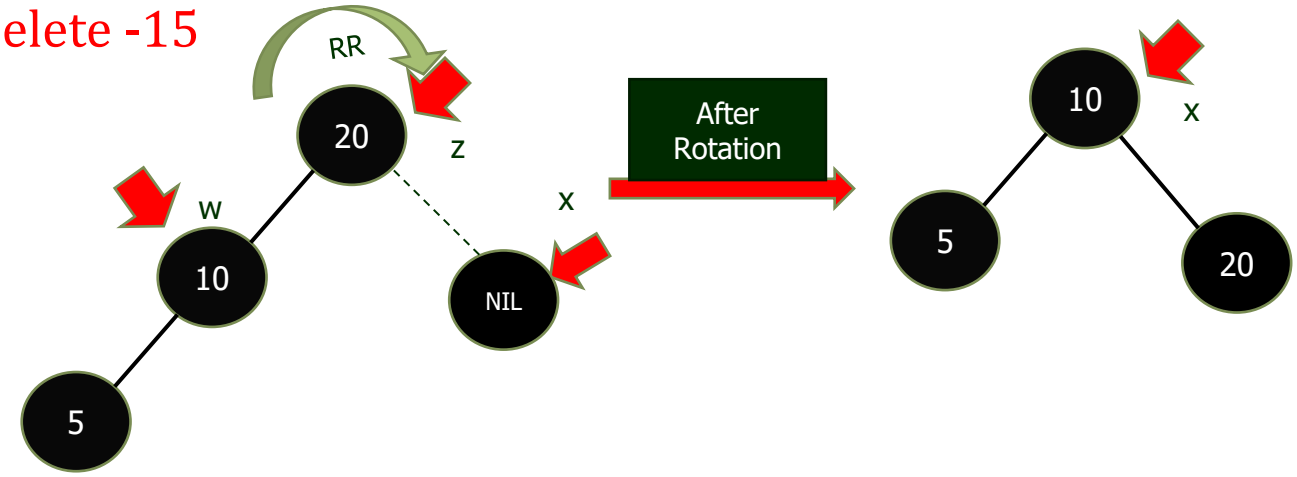
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -15



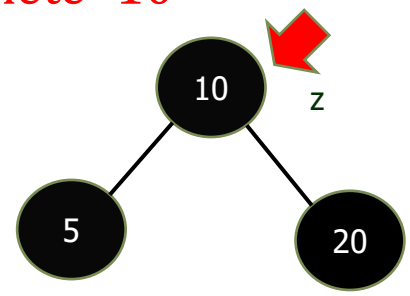
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -10



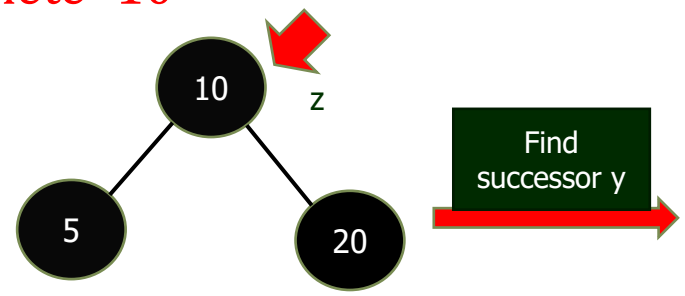
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -10



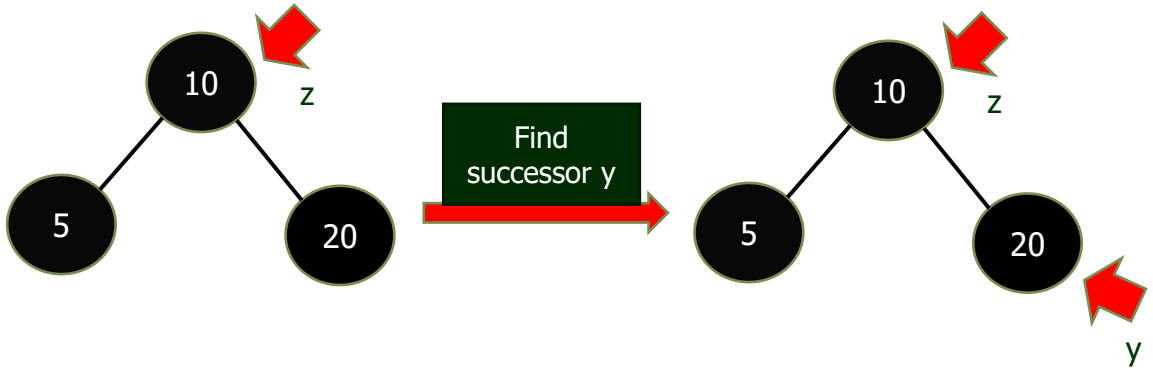
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -10



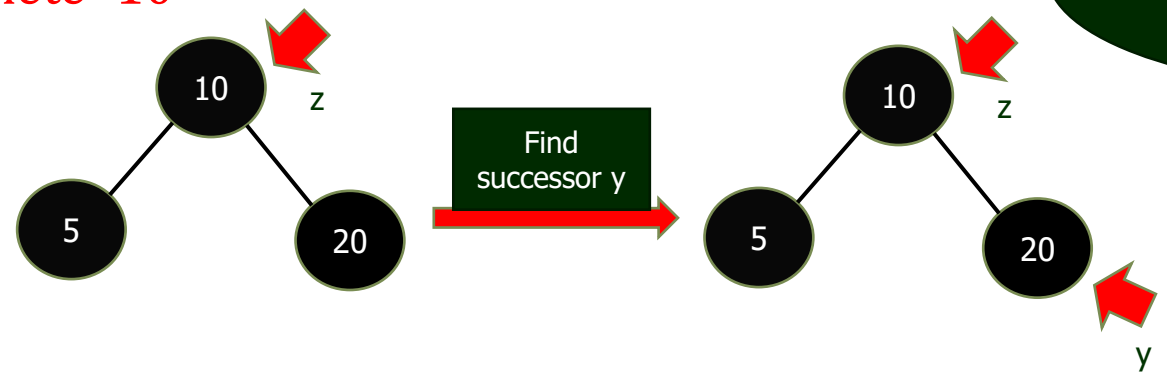
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -10



If y has no child and then simply exchange the key of y with key of z and delete y

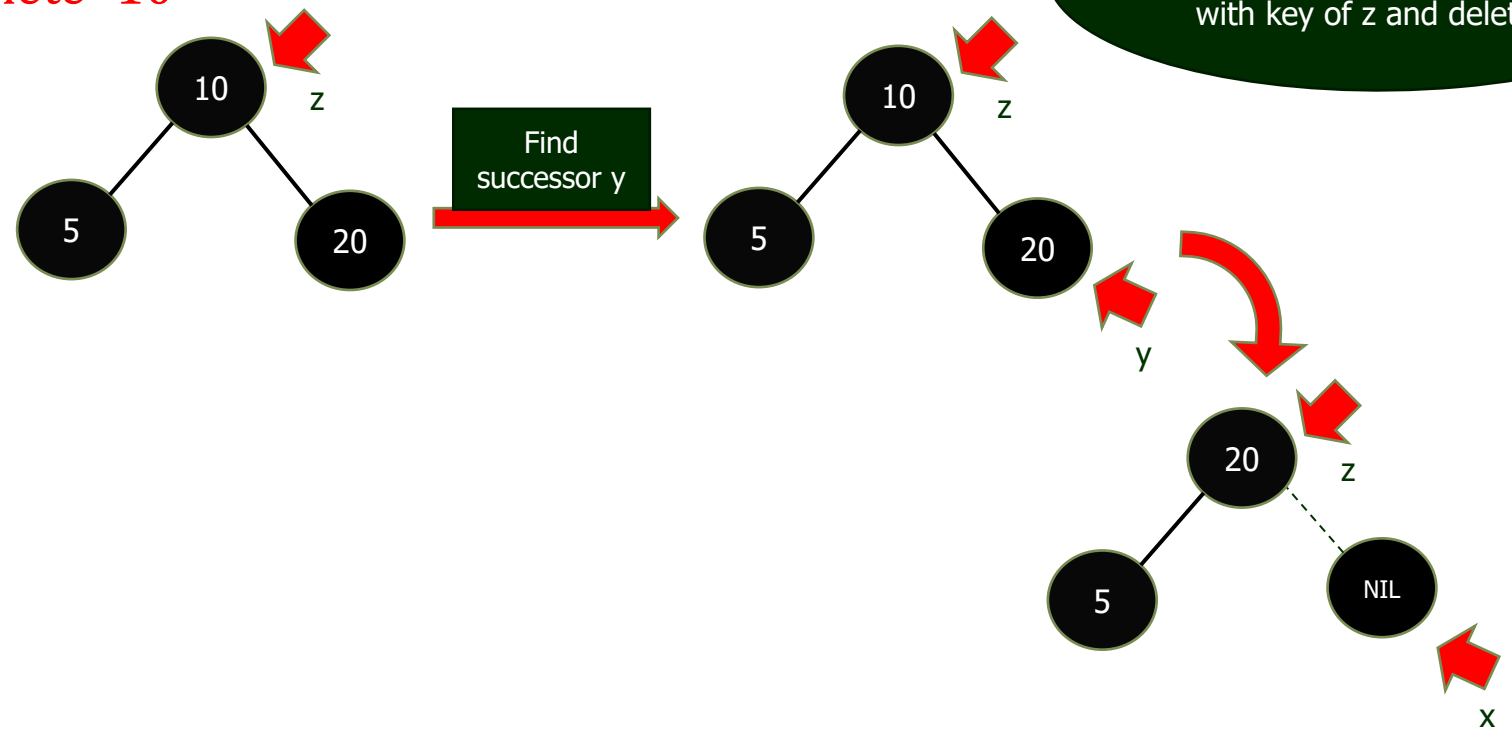
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -10



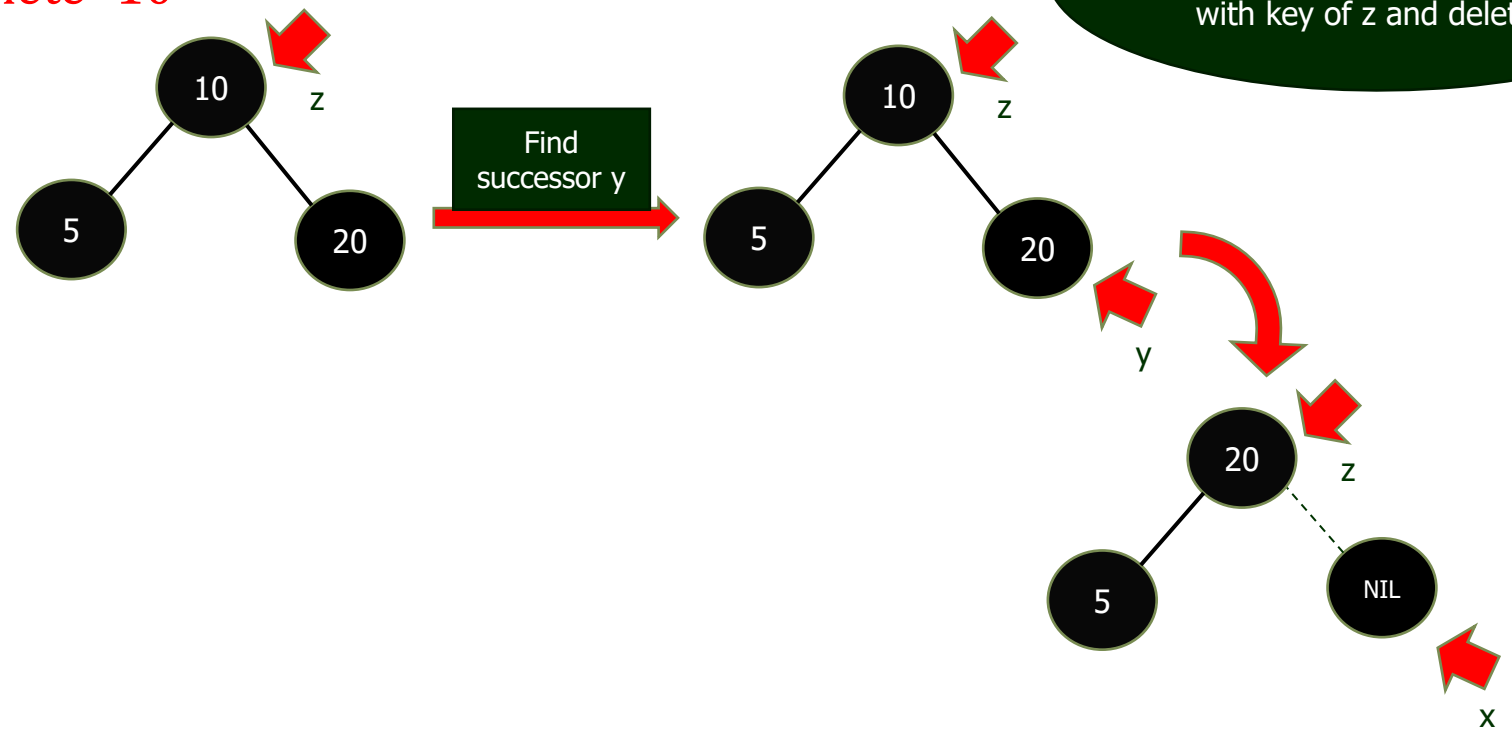
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -10



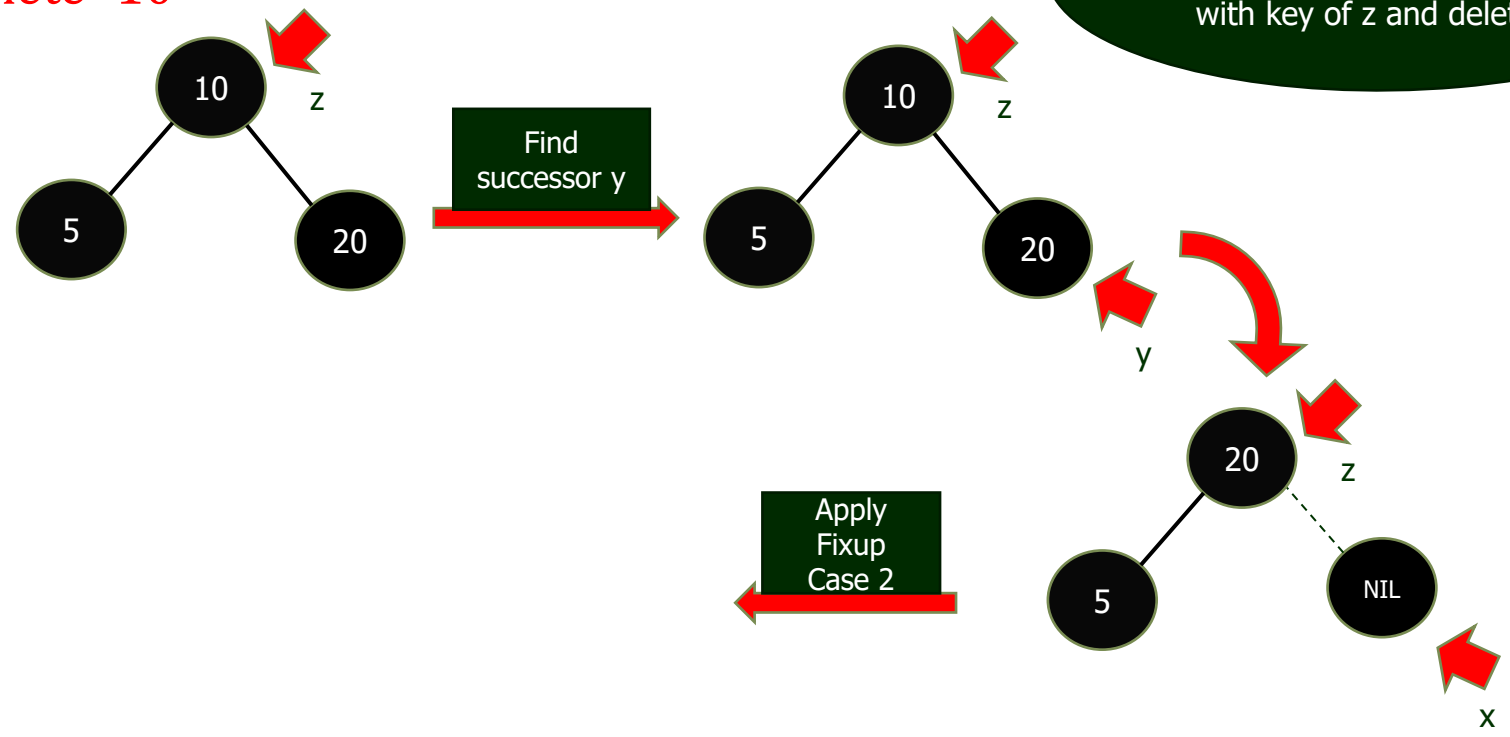
Red Black Tree

Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -10



Red Black Tree

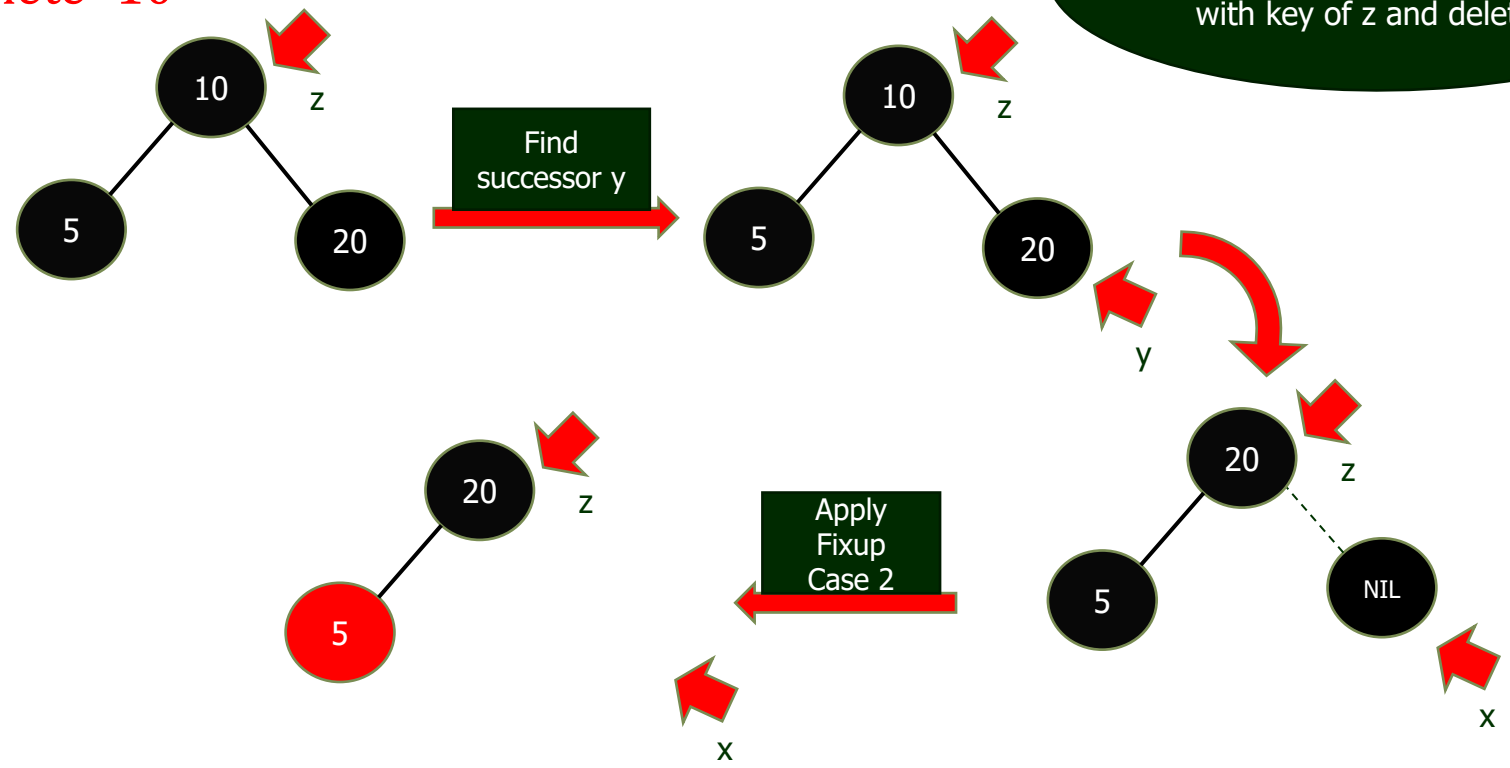
Insertion (Example 2):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <30, 25, 15 & 10>

Delete -10

If y has no child and then simply exchange the key of y with key of z and delete y

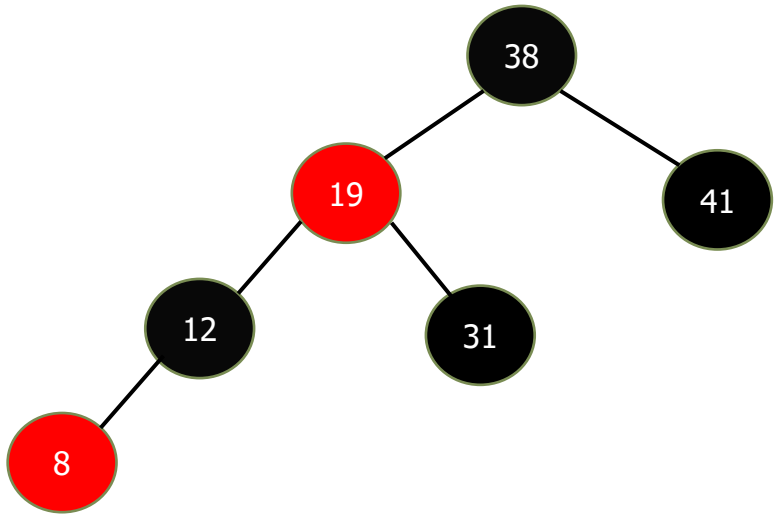


Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>



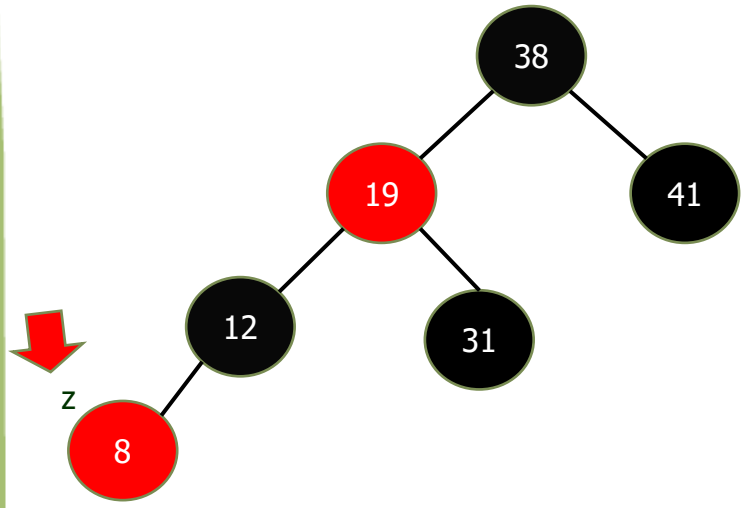
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -8



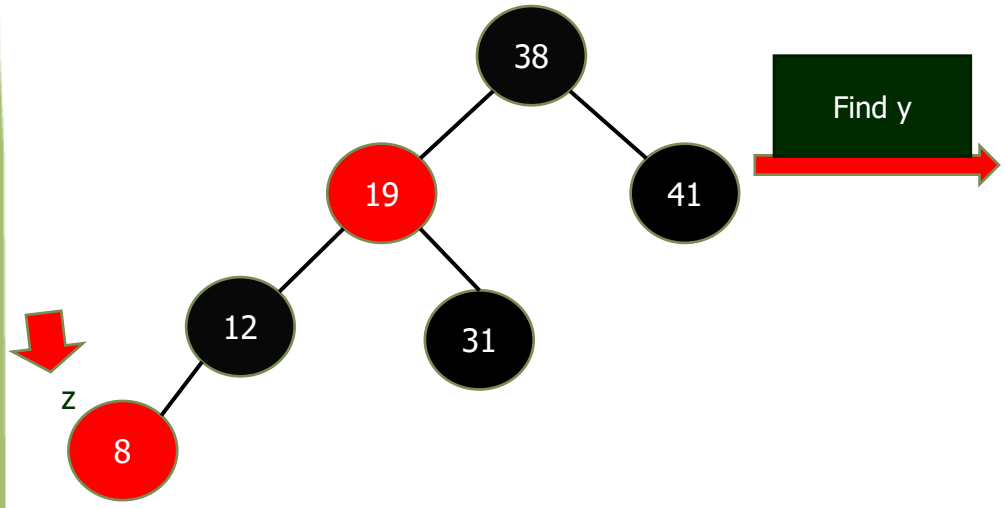
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -8



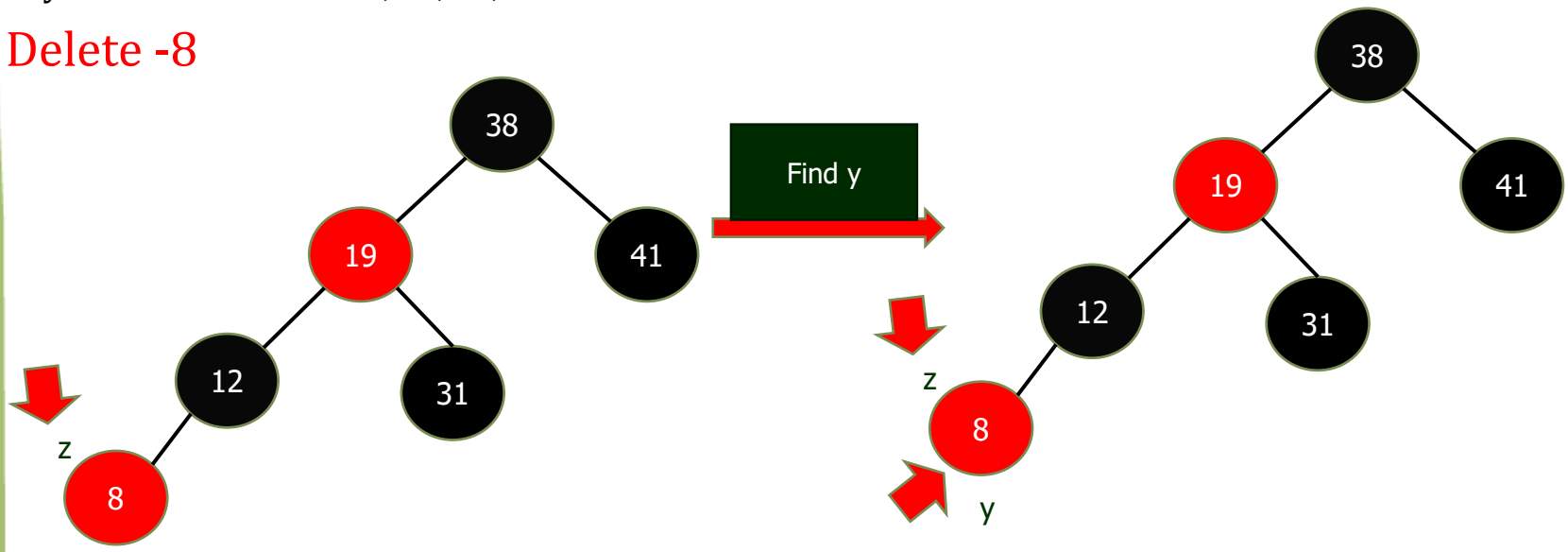
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -8



Red Black Tree

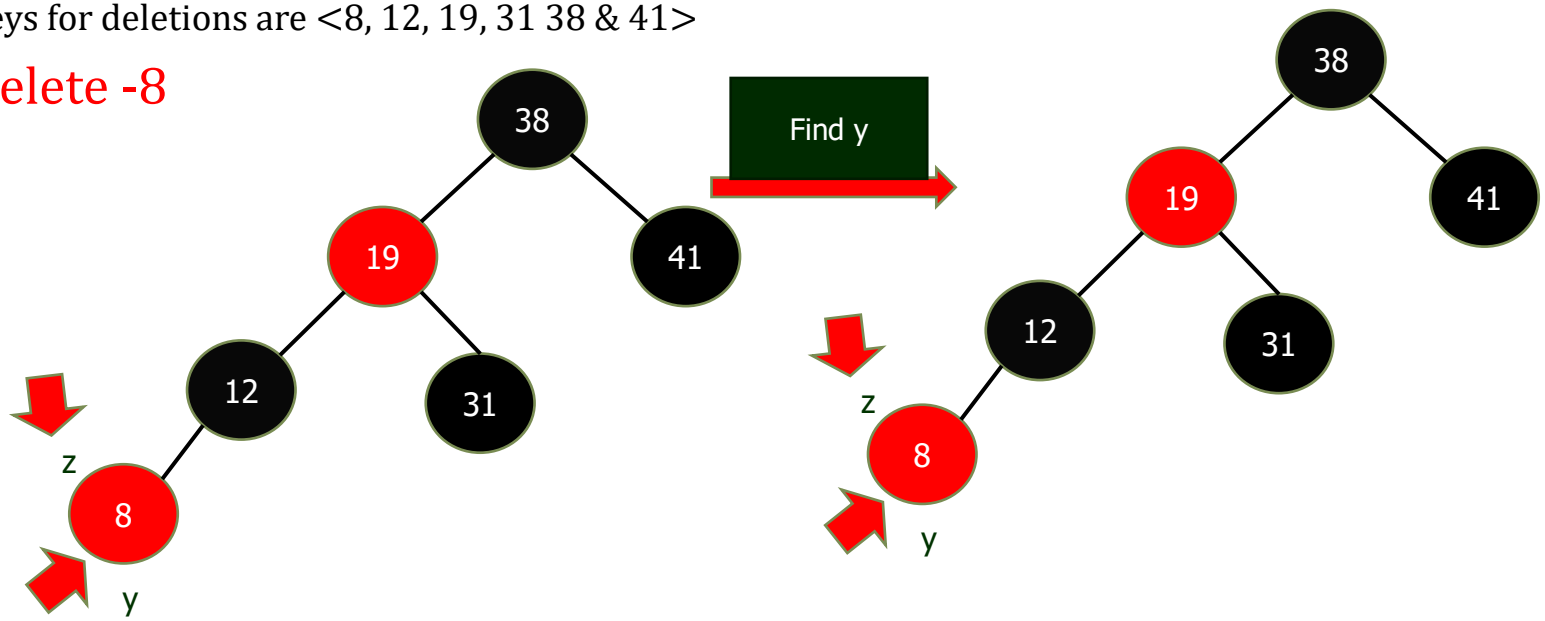
If z has no child the y=z simply delete z

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -8



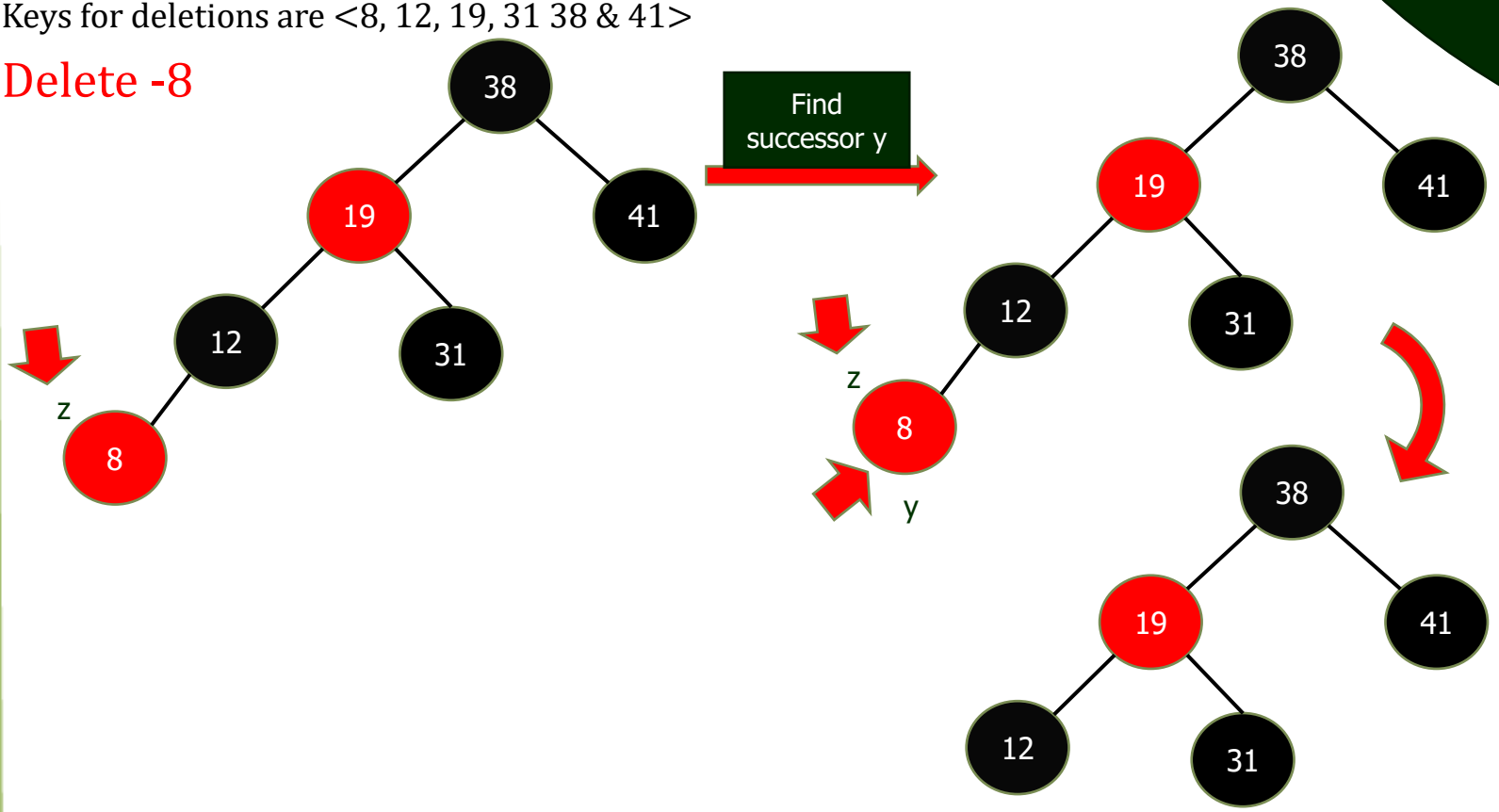
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given ke
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -8



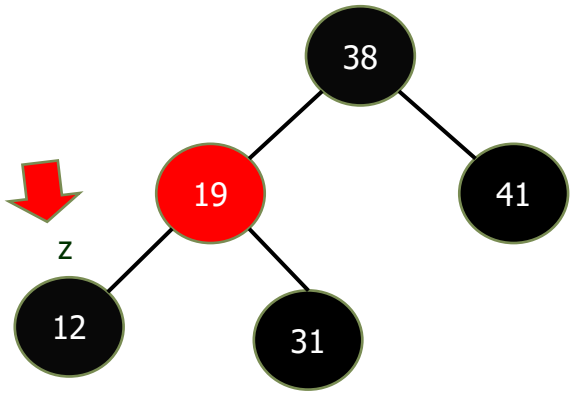
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -12



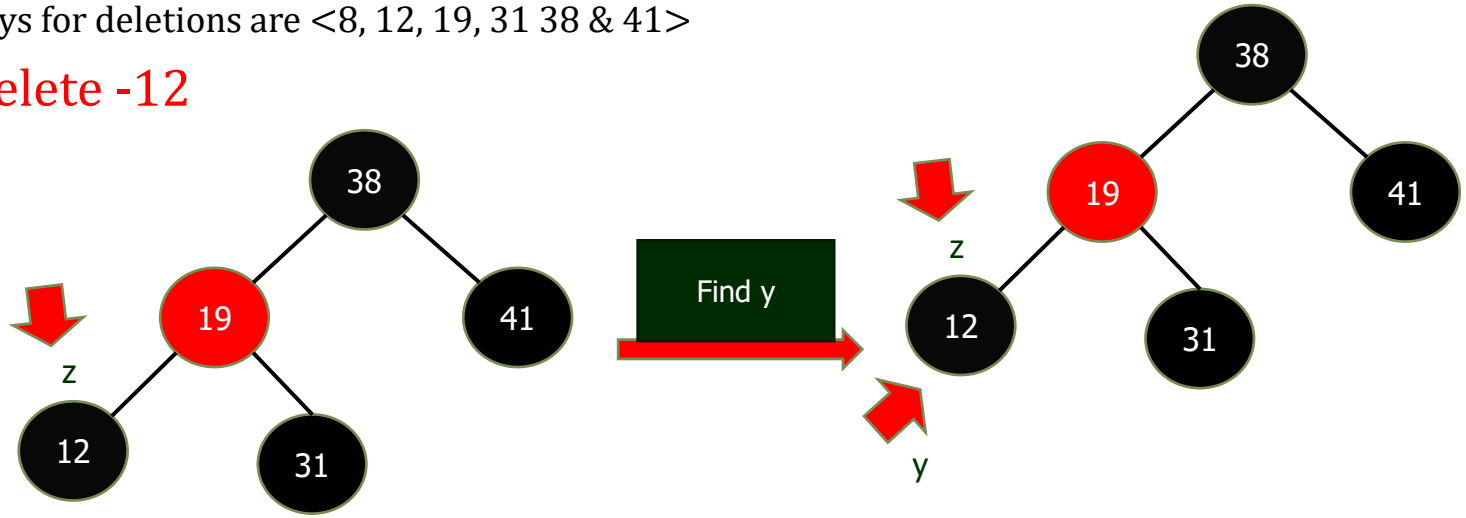
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -12



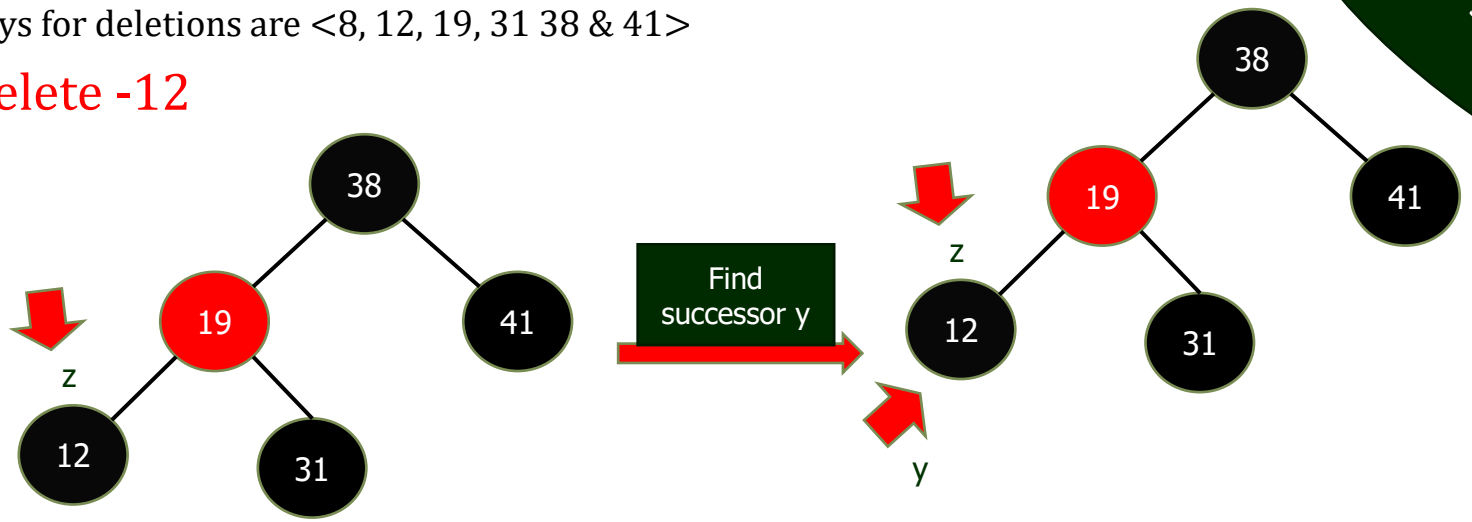
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the give
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -12



If z has no child the y=z and
simply delete z

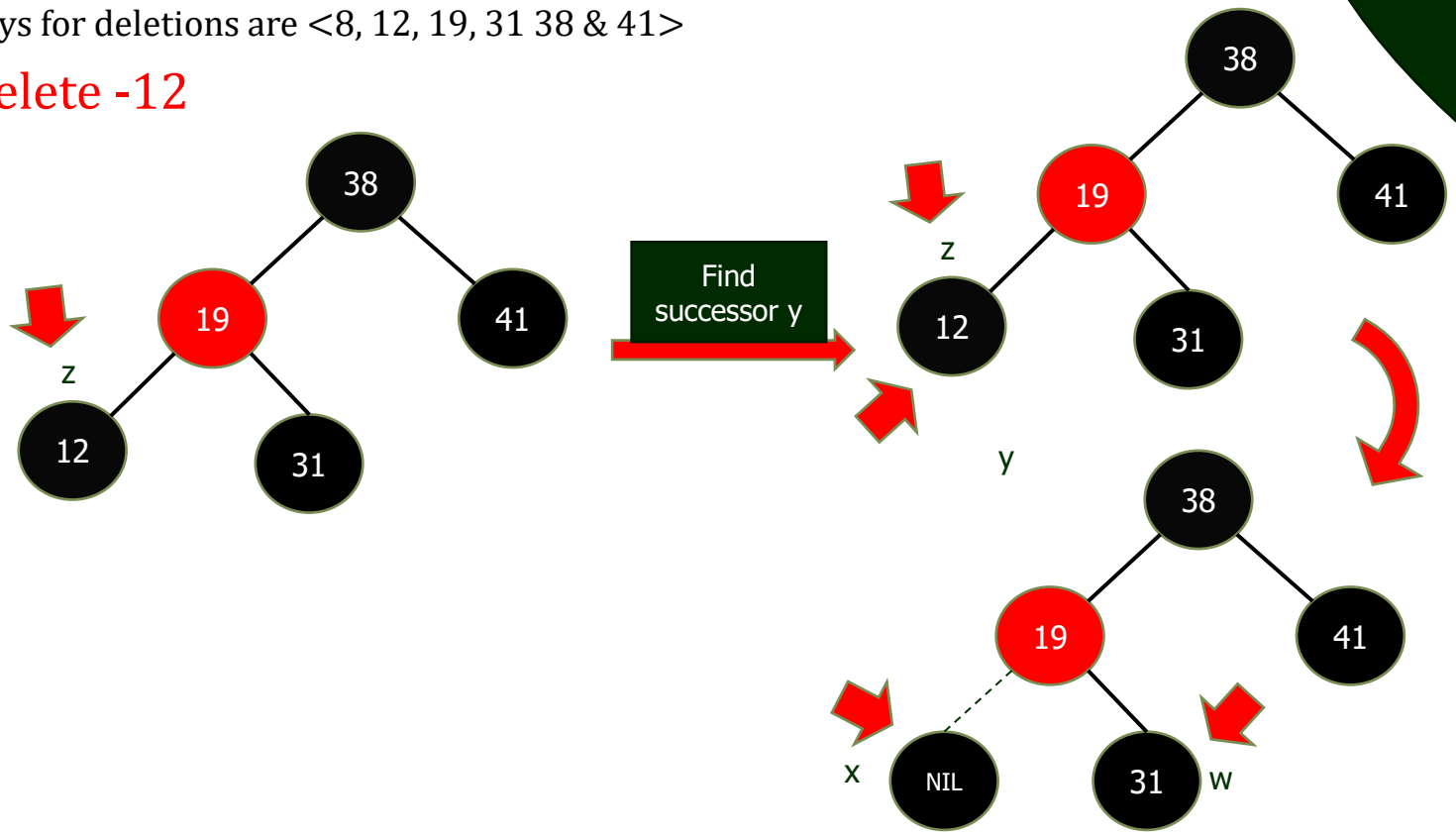
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -12



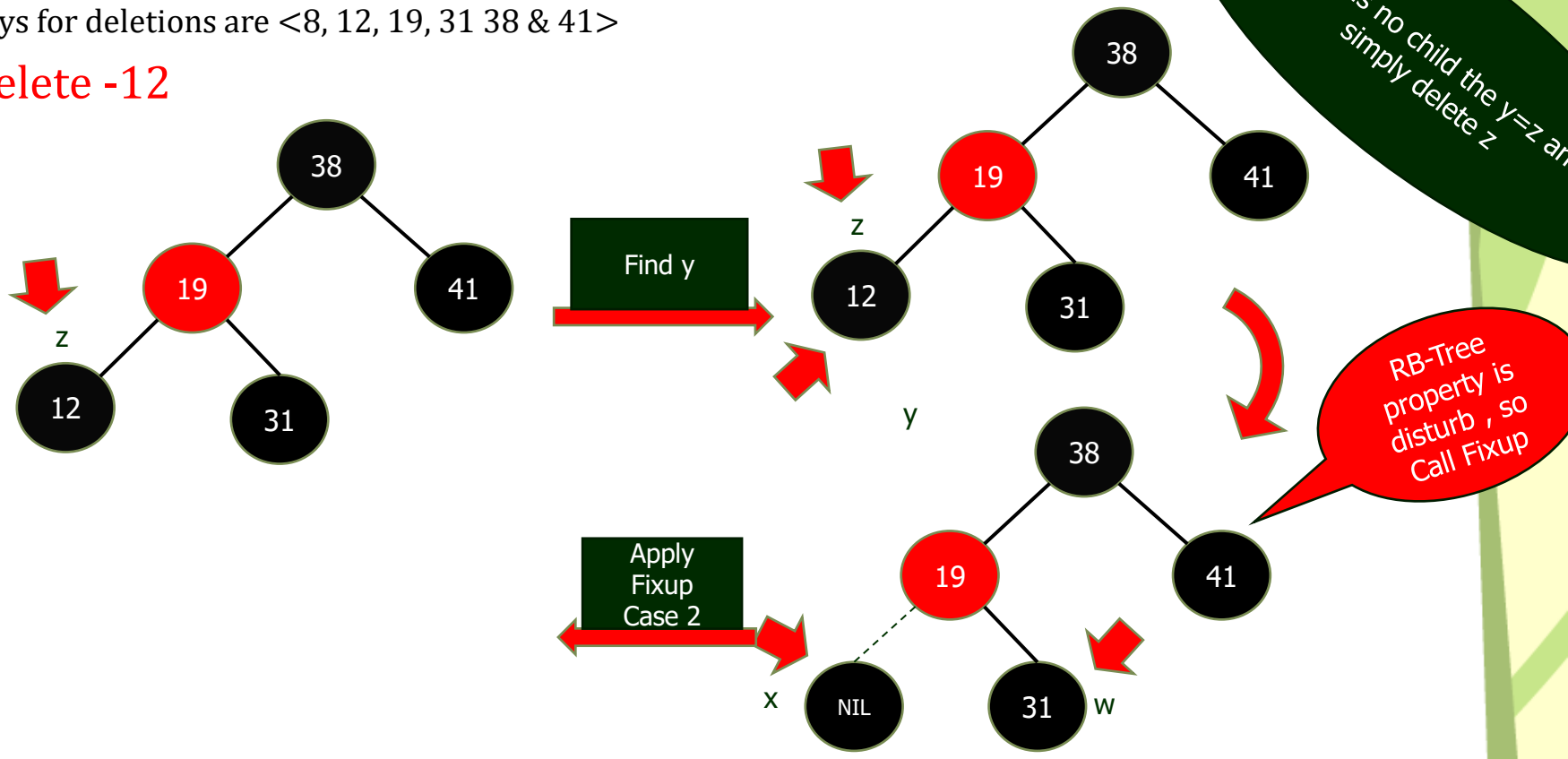
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -12



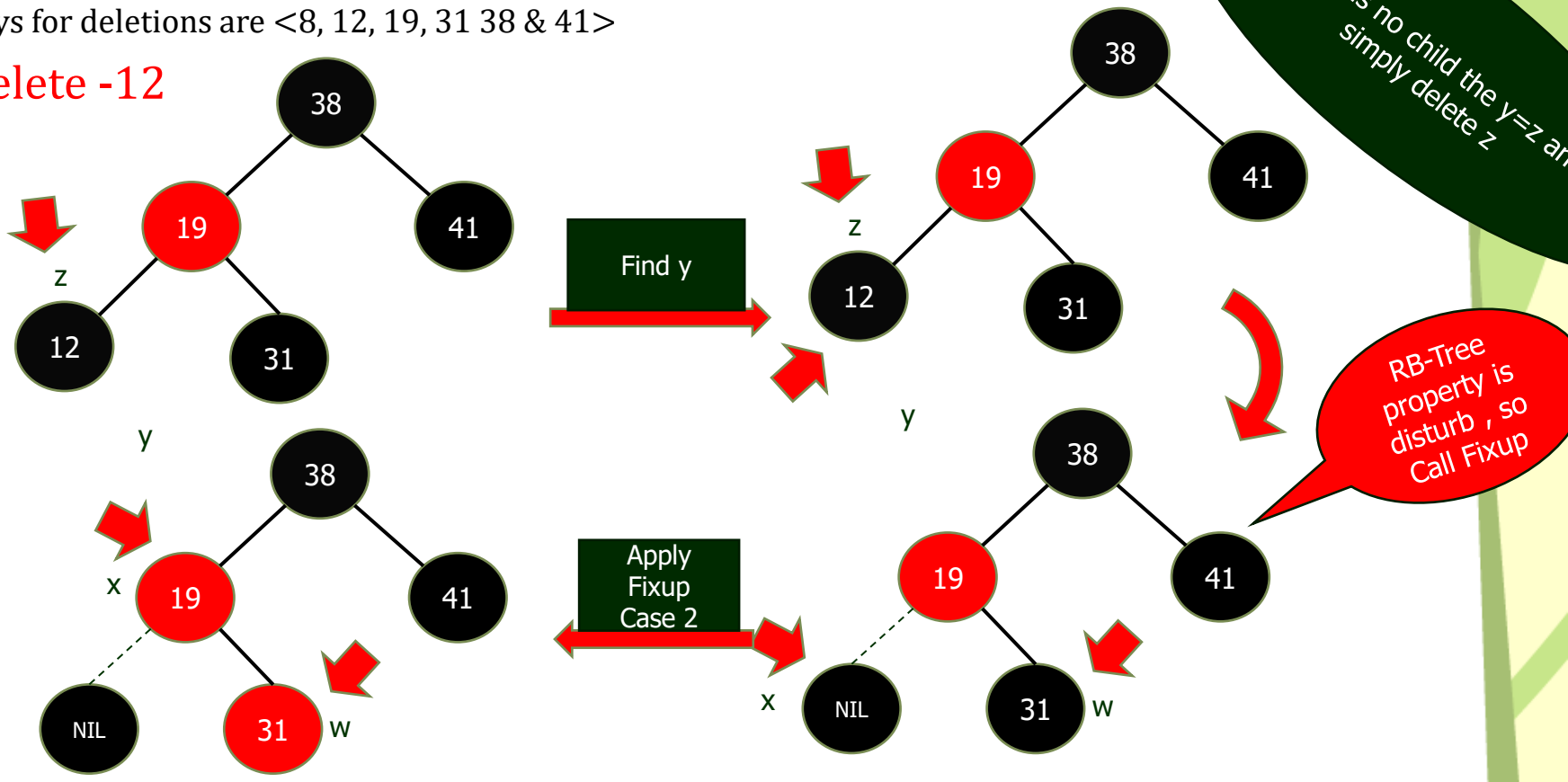
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -12



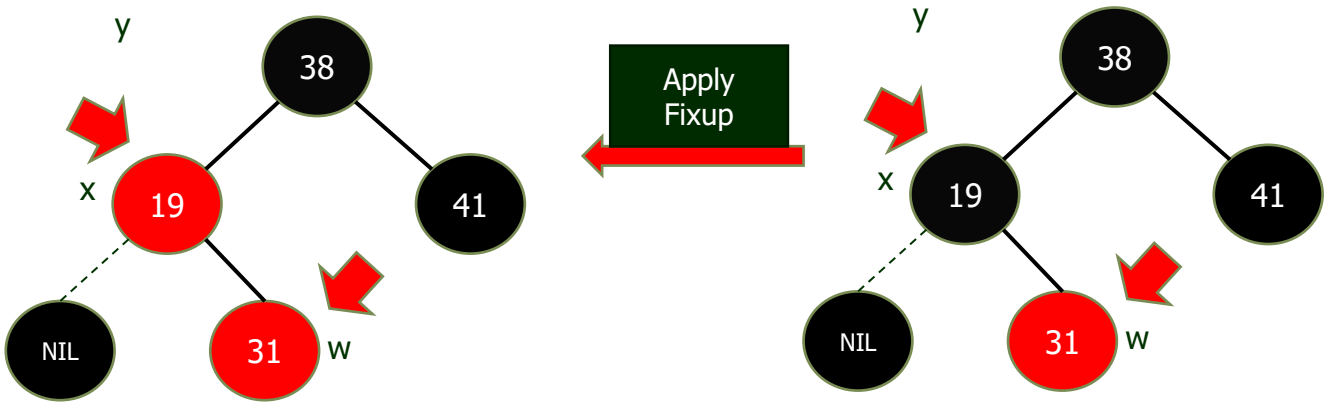
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -12



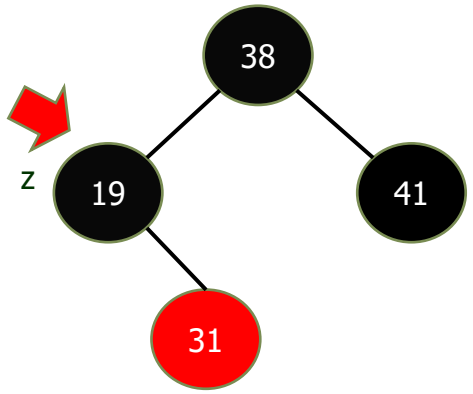
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -19



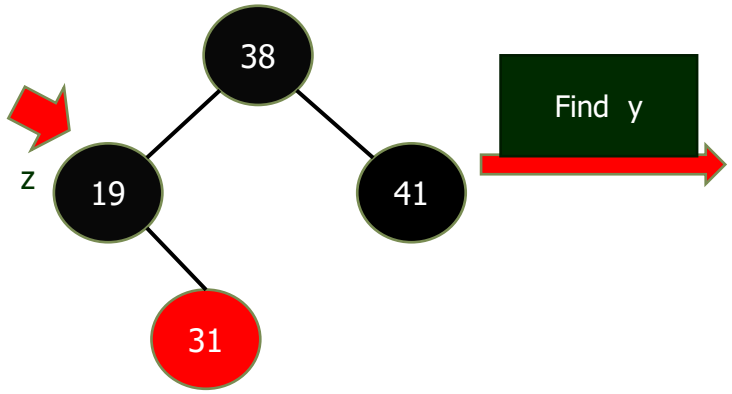
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -19



Red Black Tree

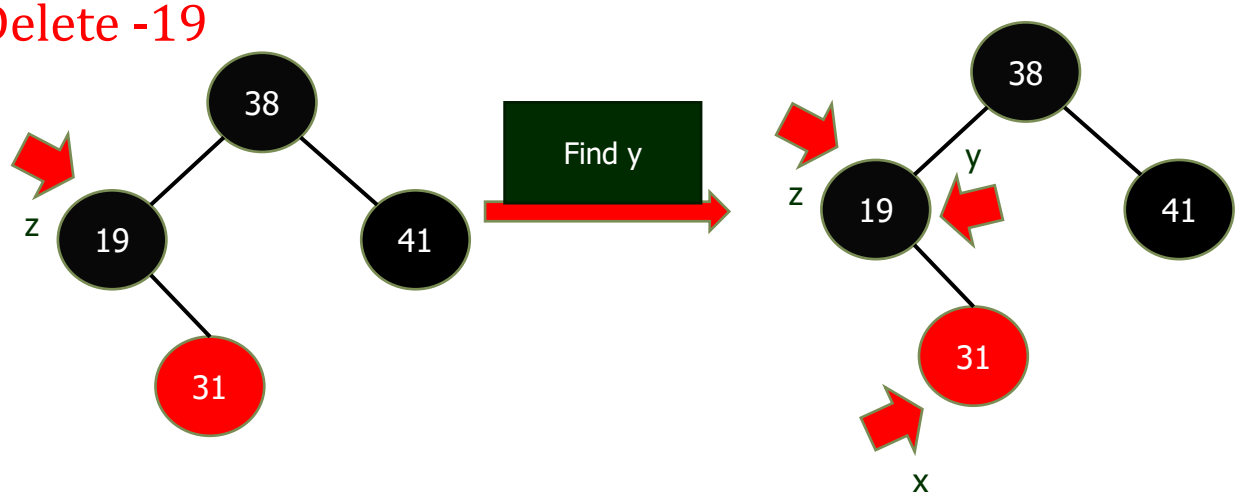
Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

If z has one child then y=z and apply BST Deletion

Delete -19



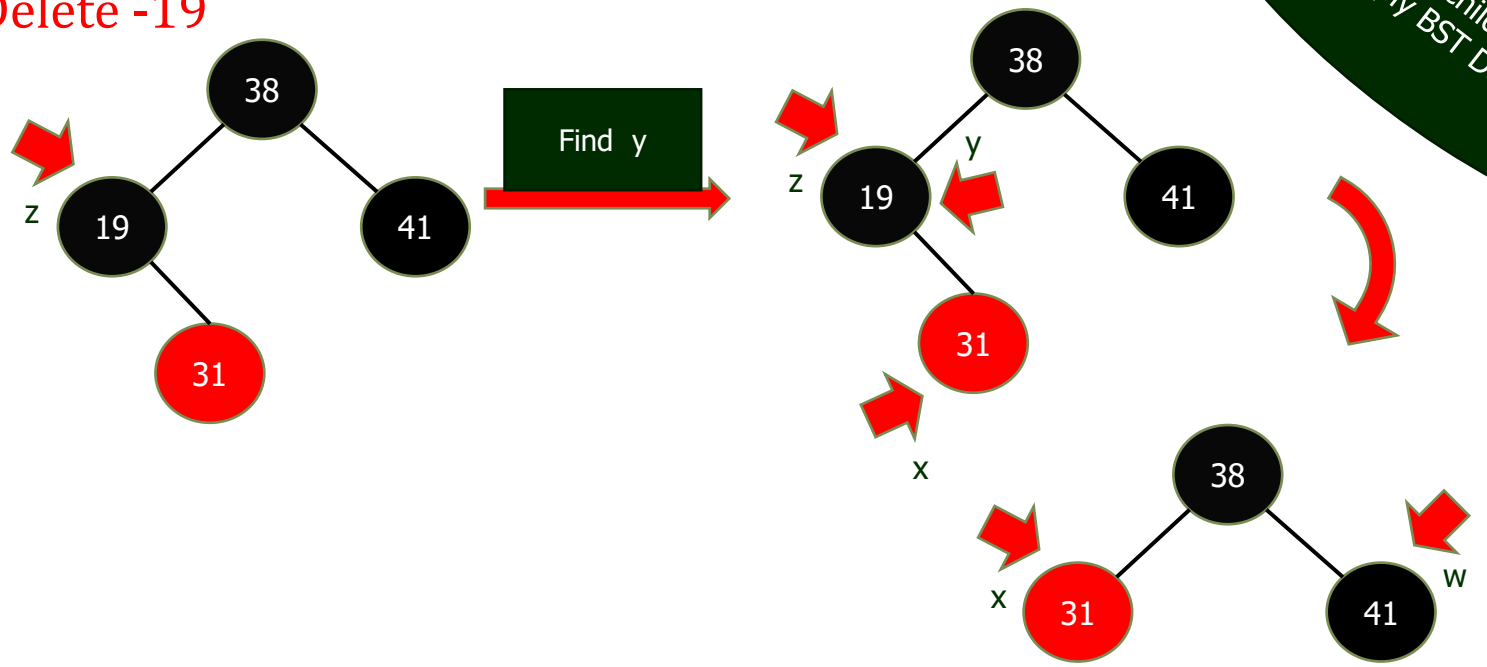
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -19



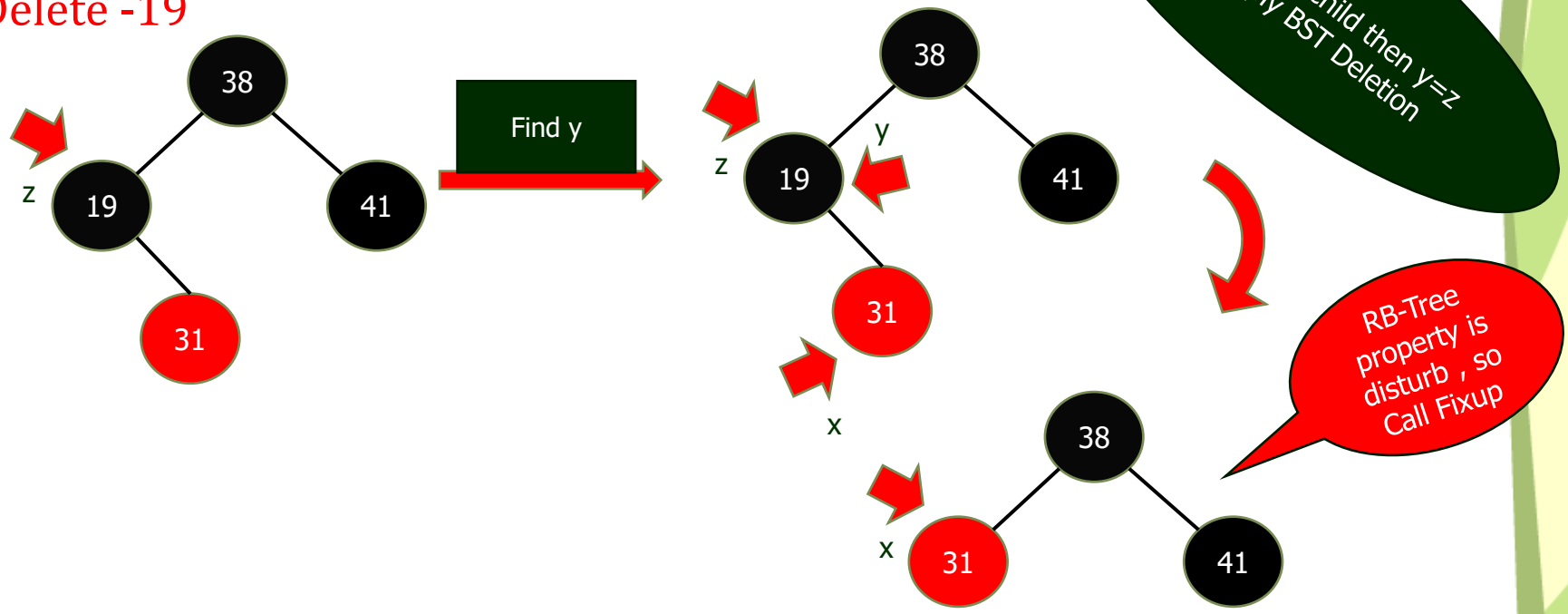
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -19



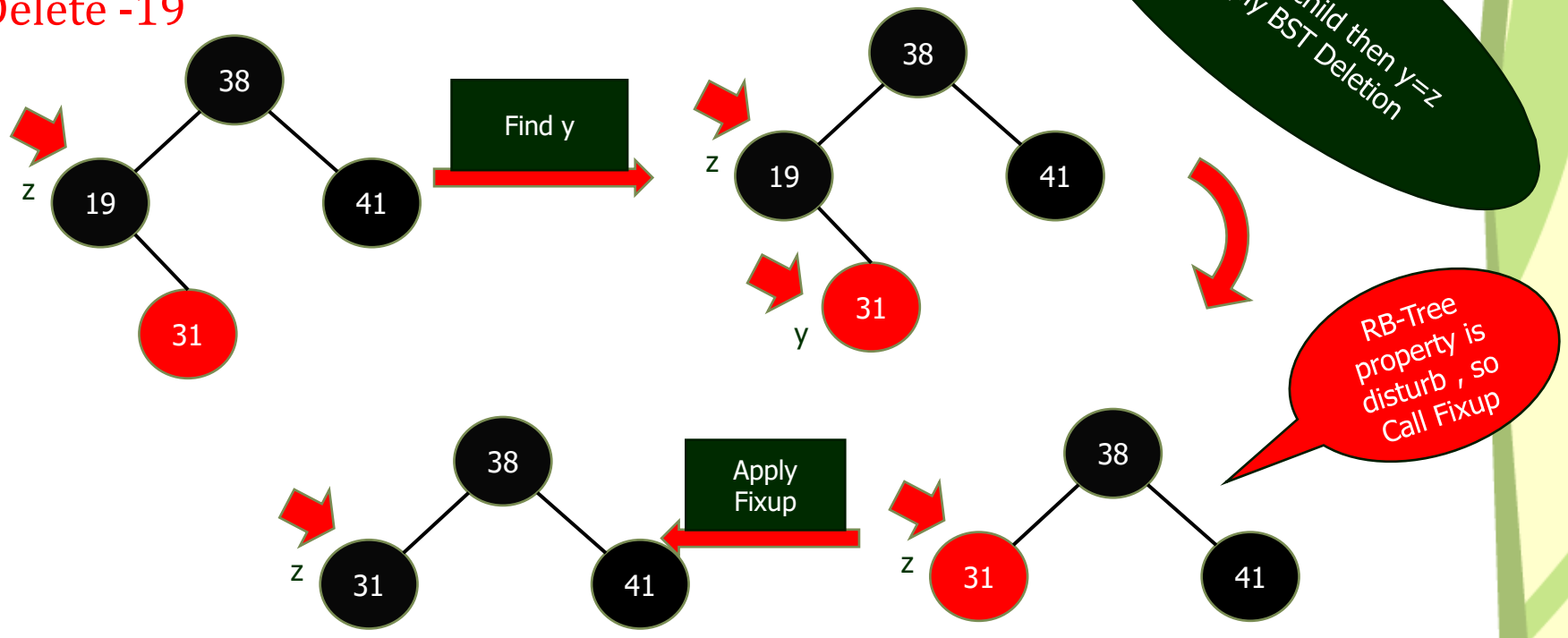
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -19



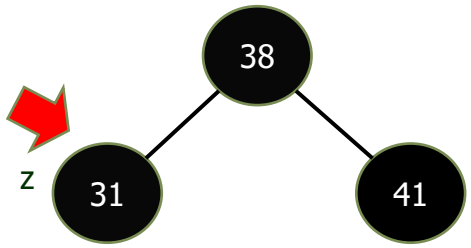
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -31



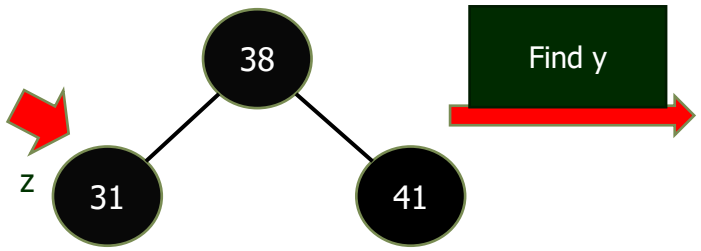
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -31



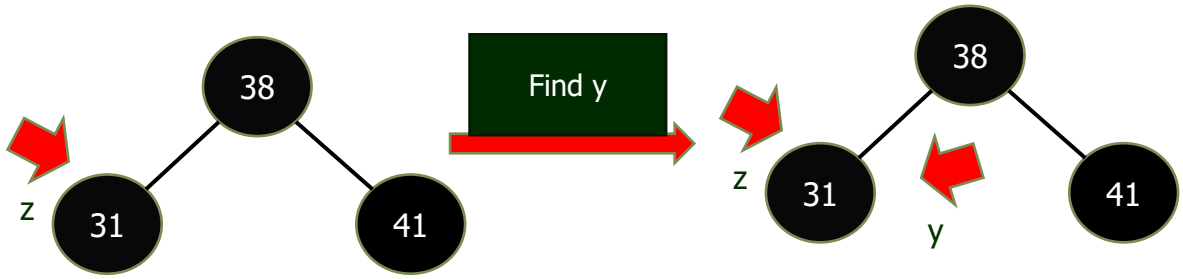
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -31



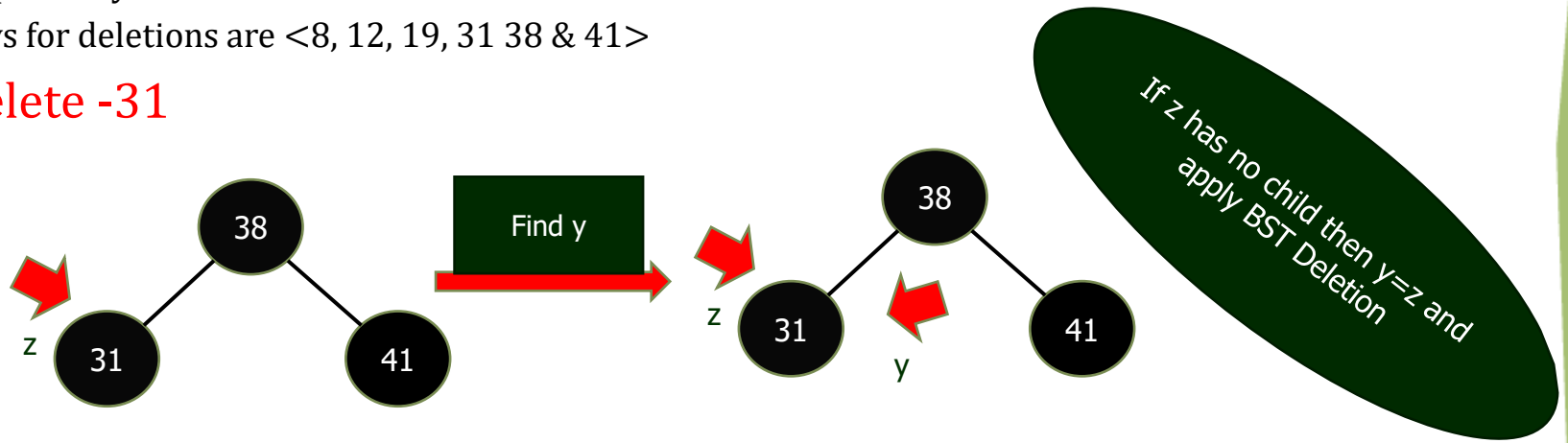
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -31



Red Black Tree

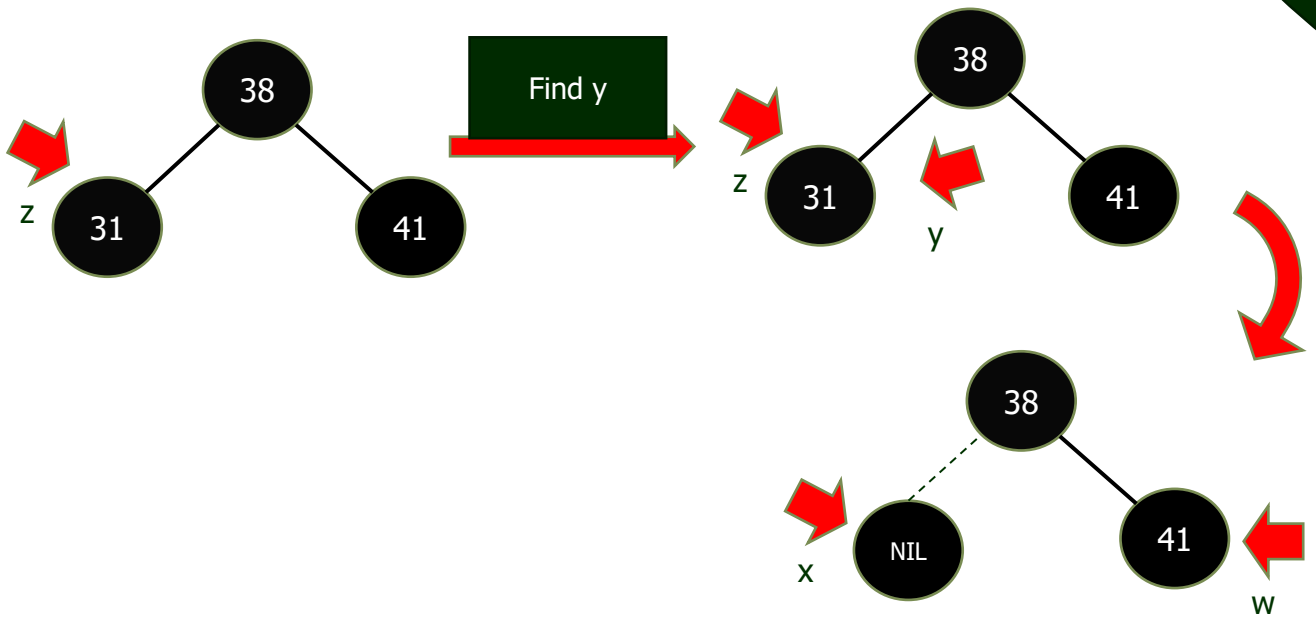
Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

If z has no child then y=z and apply BST Deletion

Delete -31



Red Black Tree

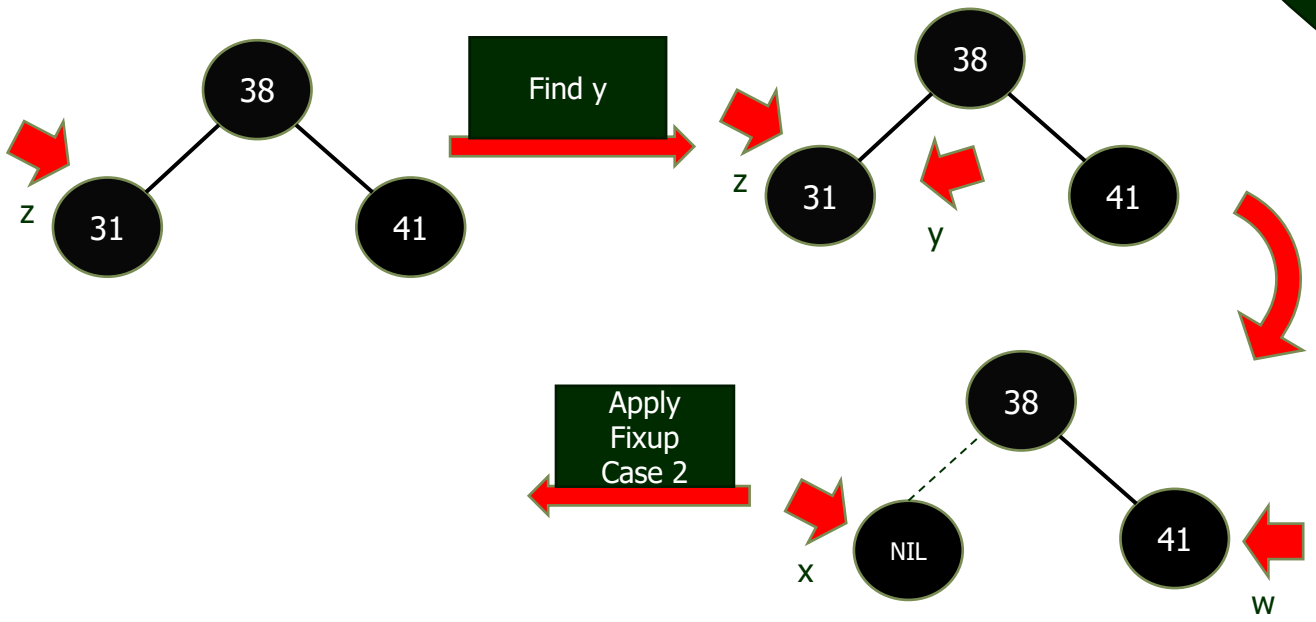
Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

If z has no child then y=z and apply BST Deletion

Delete -31



Red Black Tree

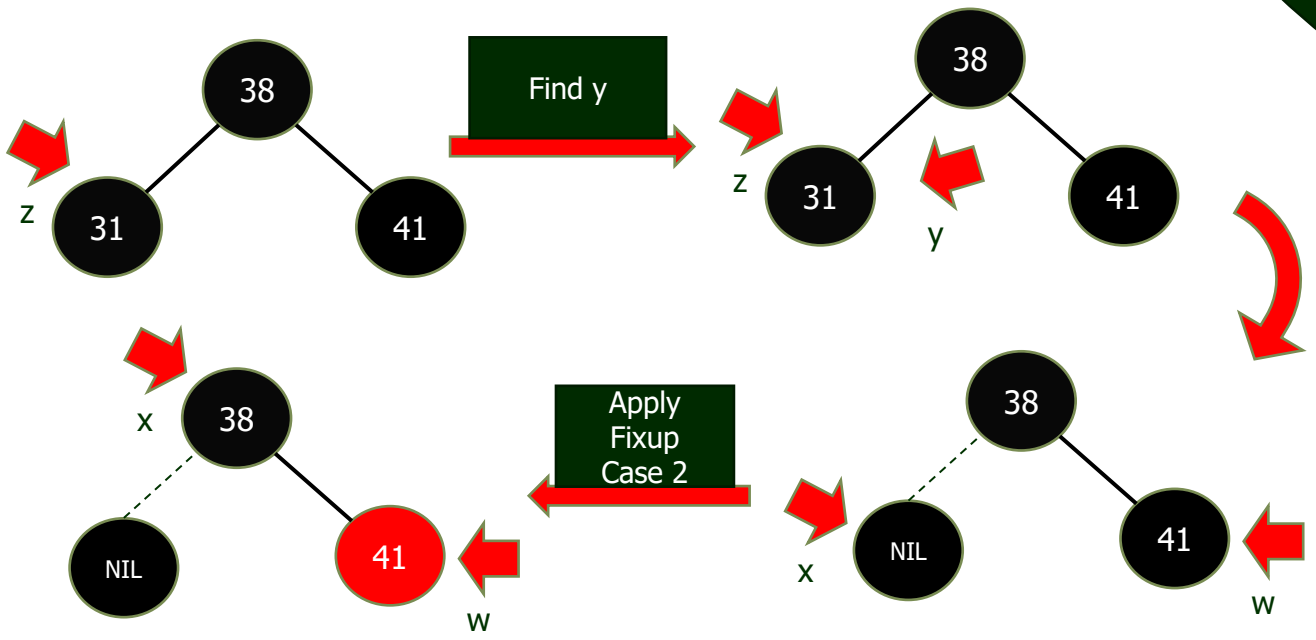
Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on
respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

If z has no child then y=z and apply BST Deletion

Delete -31



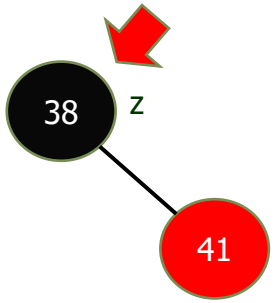
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -38



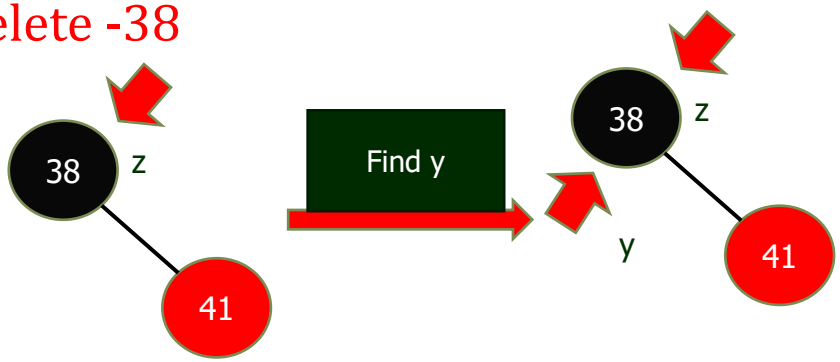
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -38



If z has no child then y=z and apply BST Deletion

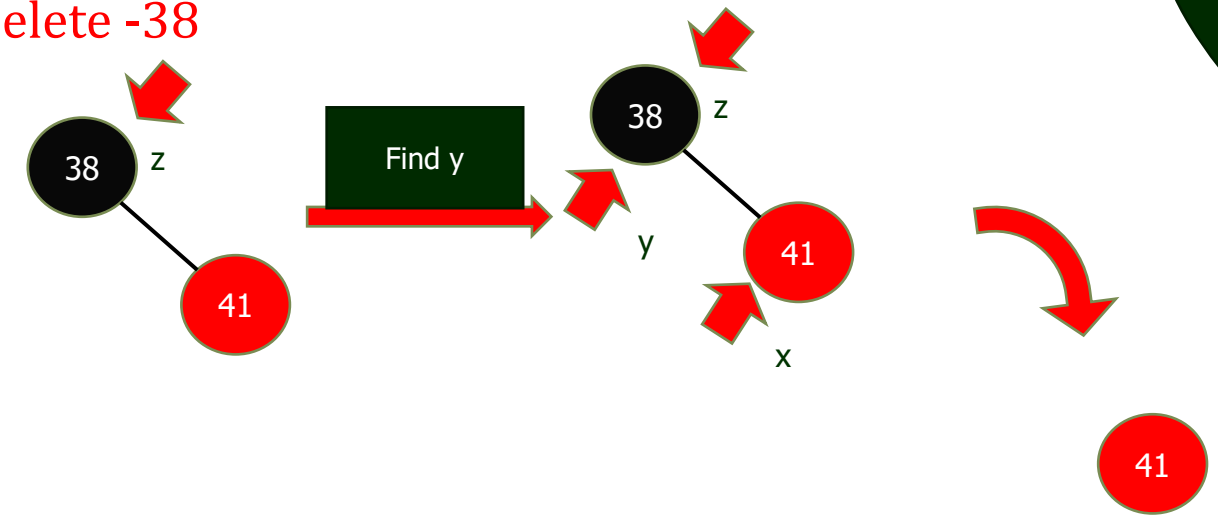
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -38



If z has no child then y=z and apply BST Deletion

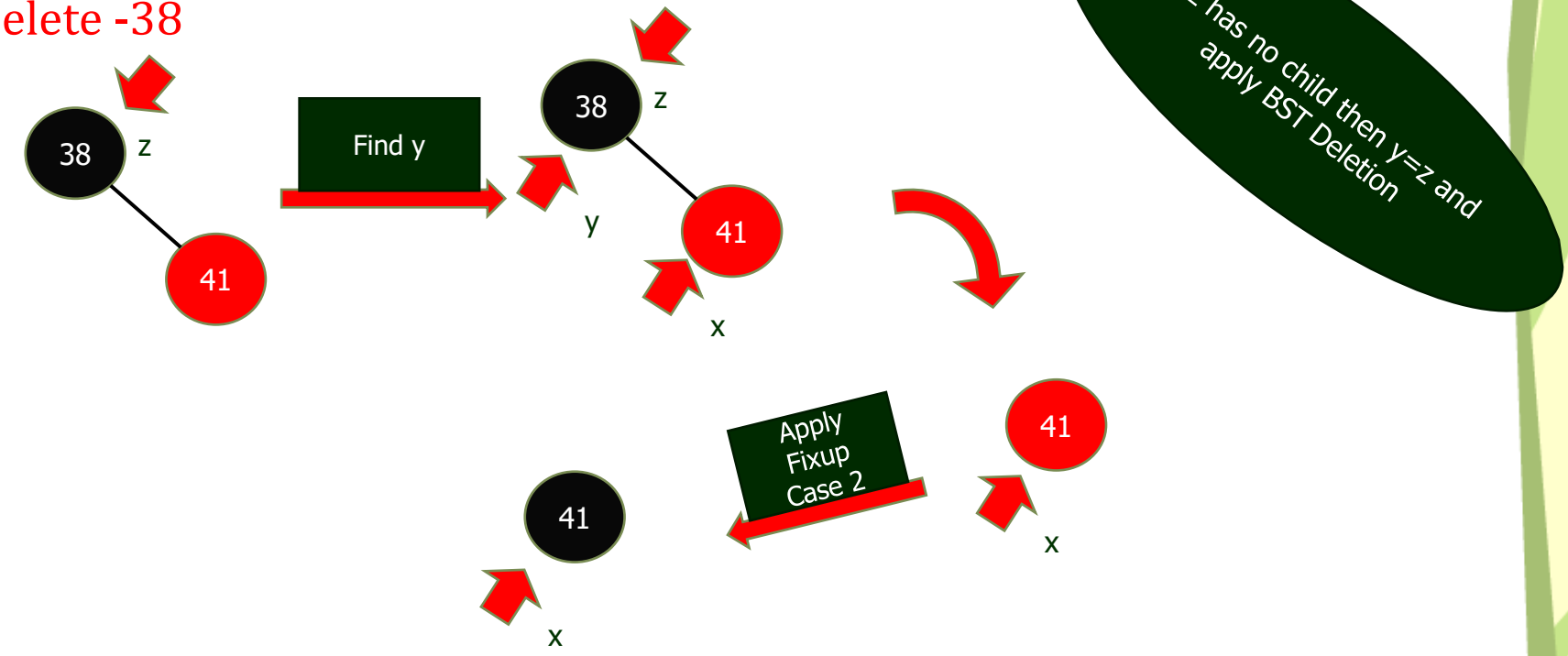
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -38



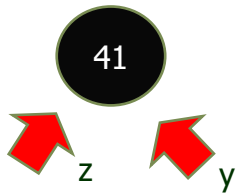
Red Black Tree

Insertion (Example 3):

With the help of following Red-black tree perform the deletion operation on the given keys respectively.

Keys for deletions are <8, 12, 19, 31 38 & 41>

Delete -41



If z has no child then y=z and apply BST Deletion

Tree is Empty

Red Black Tree

Lemma

A red-black tree with n internal nodes has $\boldsymbol{height \leq 2\lg(n+1)}$.

Proof:

The Proof is based on two number of claims:

- Claim 1: Any node with height h has black-height $\geq h/2$.
- Claim 2: The subtree rooted at any node x contains $\geq 2^{bh(x)} - 1$ internal nodes.

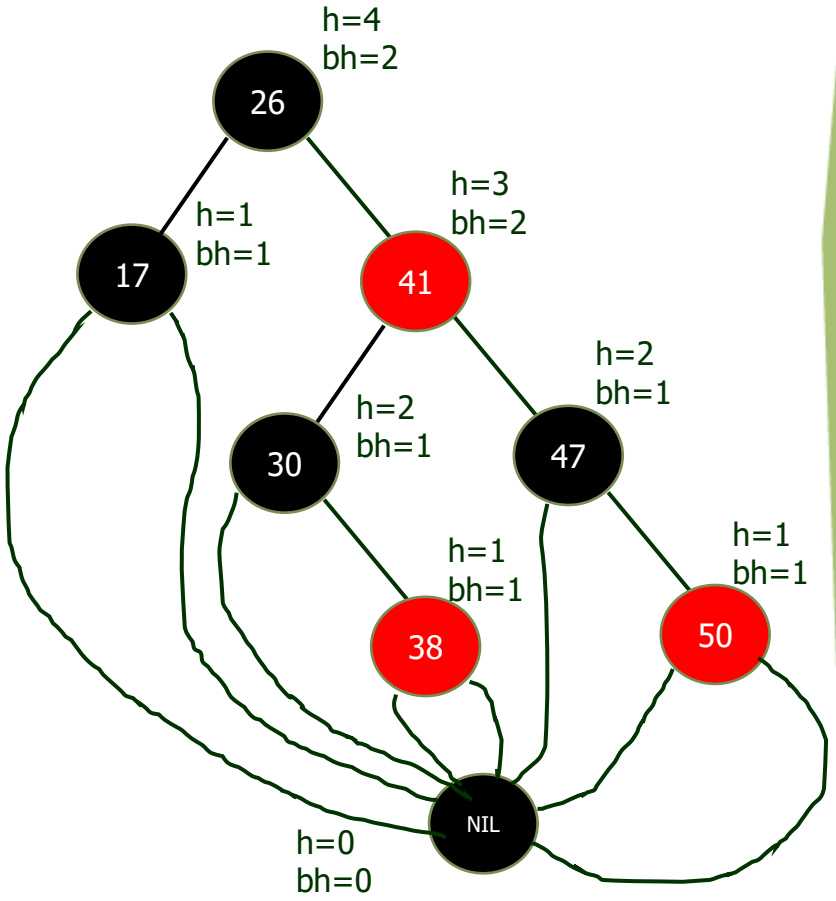
Red Black Tree

- Claim 1: Any node with height h has black-height ≥ h/2.

According to property 4(i.e. If a node is red, then both its children are black) , at least half of the nodes on any simple path from root to leaf node, excluding root node must be black.

Hence, any node(x) with height h has bh(x) ≥ h/2 (i.e. h/2 number of black node.)

proved



Red Black Tree

Claim 2:

The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal nodes (i.e. the nodes who bearing keys only).

Proof by induction method on height of x.

Base case:

When x is a leaf node, h(x)=0 and bh(x)=0.

Hence the internal node of x is $= 2^{bh(x)} - 1$

$= 2^0 - 1$

=1-1=0

Red Black Tree

Induction hypothesis:

Consider a node x that has a positive height and is an internal node with two children of left and right.

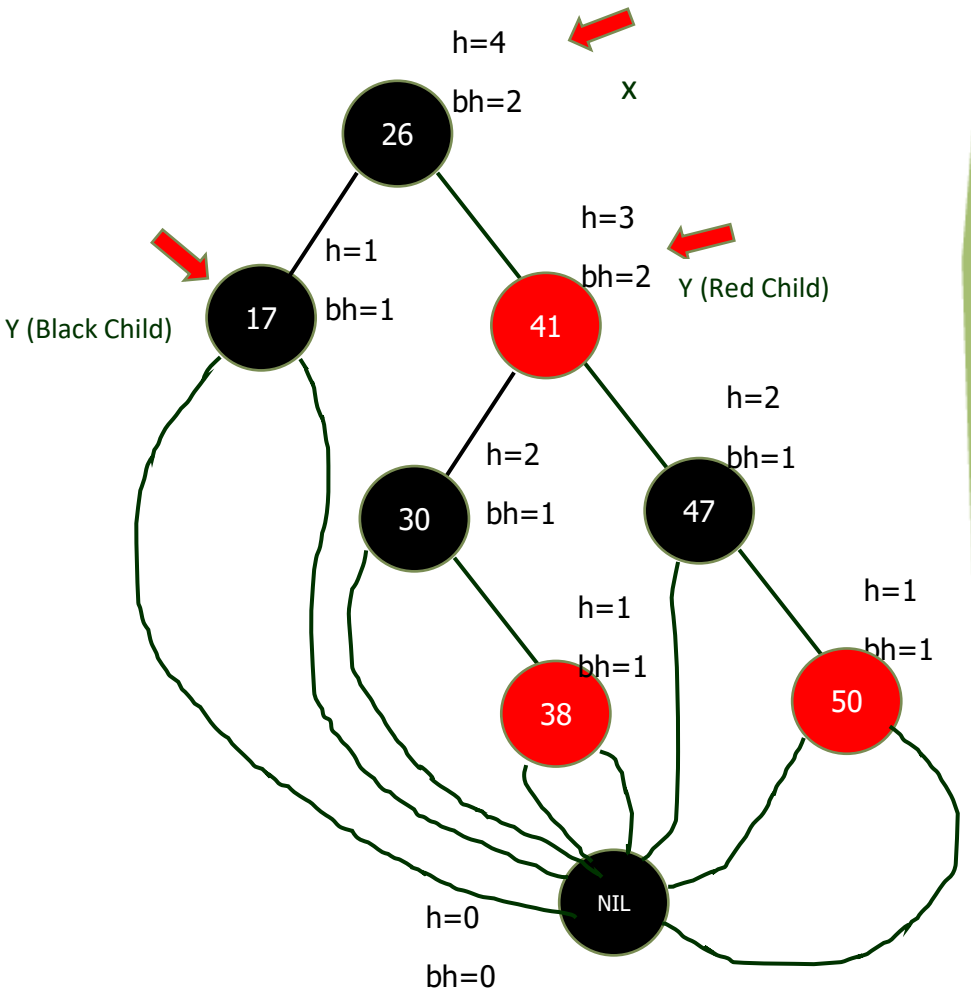
Each children has a black height of either bh(x) or bh(x)-1 depending on weather it's color is red or black respectively.

For Example :

Let bh(x)=b,

Then bh(y)=b if y is a red child of x

bh(y)=b-1 if y is a black child of x.



Red Black Tree

Since ,

$$h\big(child(x)\big) < h(x)$$

By using the inductive hypothesis, we can conclude that each child has at least $2^{bh(x)-1} - 1$ internal node. Thus , the subtree rooted at x contain at least

$$(2^{bh(x)-1}-1) + (2^{bh(x)-1}-1) +1= 2^{bh(x)} - 1 \text{ internal nodes.}$$

Which proves the claim 2.

So as per the definition of claim 1 , the bh(root) must be at least $h/2$.

Hence $\Longrightarrow n \geq 2^{bh(x)} - 1$

$\Longrightarrow n \geq 2^{h/2} - 1$ (as bh(x)=h/2) [claim 1]

$\Longrightarrow n+1 \geq 2^{h/2}$

Apply log both side

$\Longrightarrow \log(n+1) \geq \log 2^{h/2}$

$\Longrightarrow \log(n+1) \geq h/2 \log 2$

$\Longrightarrow \log(n+1) \geq h/2$

$\Longrightarrow h \leq 2 \log(n+1)$ Proved


Thank U