Red-Black Trees

Red-Black Trees

- Red-Black Tree (RB-Tree) is a self balancing or height balancing binary search tree (BST) that performs all the operations (search, insert and delete) in logarithmic time (LogH).
- RB-Tree is a BST with additional information of color at each node
  - The color can be either RED or BLACK
- By constructing the way nodes can be colored on any path from the root to a leaf, RB-Tree ensure that no such path is more than twice as long as any other, so that the tree is approximately balanced.
- Children of leaf (internal) nodes i.e. NIL nodes are called external nodes of the tree.

Red-Black Properties

- A BST is a RB-Tree if it satisfies the following red-black properties:
  1. Every node is either red or black
  2. The root is black
  3. Every external - leaf (NIL) is black
  4. If a node is red, then both its children are black

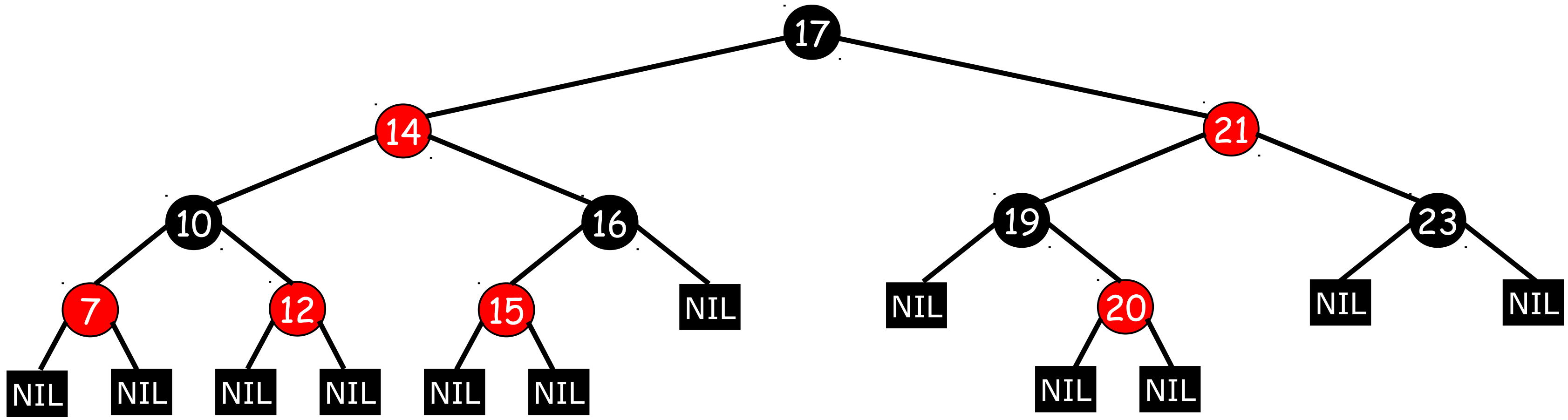
     OR

     If a node is red, then its parent is black

     OR

     A red node cannot have a red node a its child
  5. For each node, all paths from the node to descendent leaves contain same number of black nodes (black height).

RB-Tree: Example

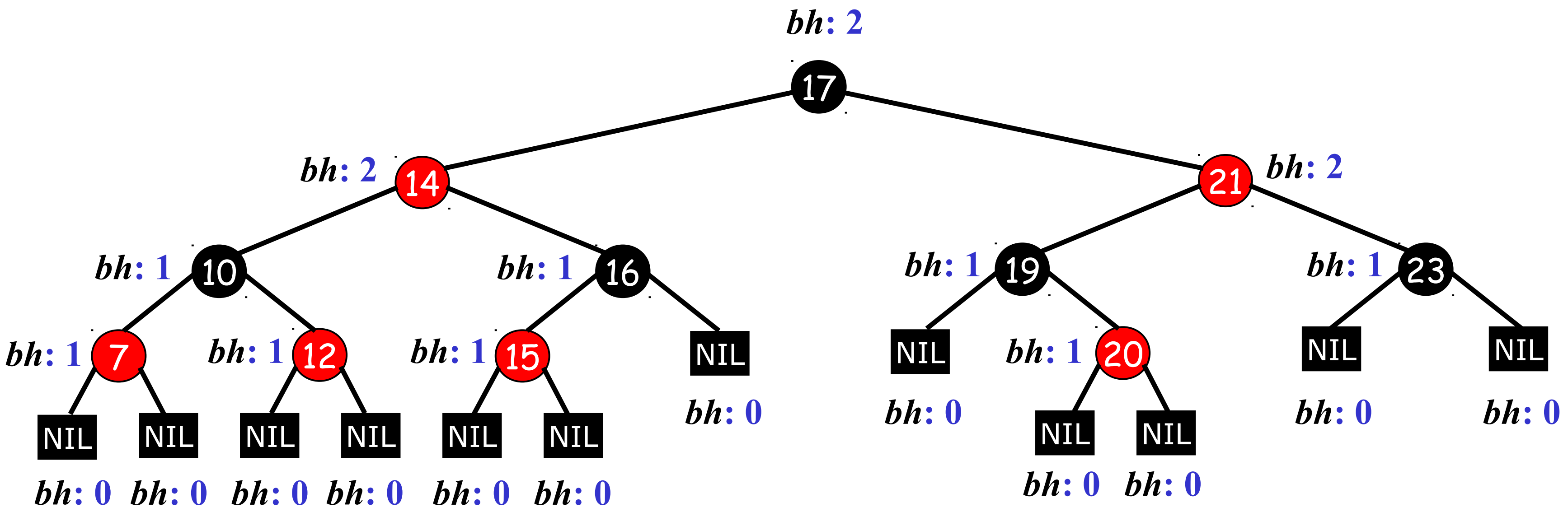


1. Every node is either red or black
2. The root is black
3. Every leaf (NIL) is black
4. If a node is red, then both its children are black
5. For each node, all paths from the node to descendent leaves contain same number of black nodes

Black-Height

- The property 5 of the red-black property is the black height of a node
- Black-height of a node:
  - The number of black nodes on any path from a node x, not including the node x, down to a leaf is called **black-height of the node x, $bh(x)$**
- Black-height of a RB-Tree:
  - Number of black nodes on any path from the root, not including the root, down to a leaf (external nodes) is called **black-height of a RB-Tree**
  - It should be noted that, the number of black nodes from root to any external node (NIL) is same

Black-Height: Example



- Black-height of tree: 2

Height of a RB-Tree

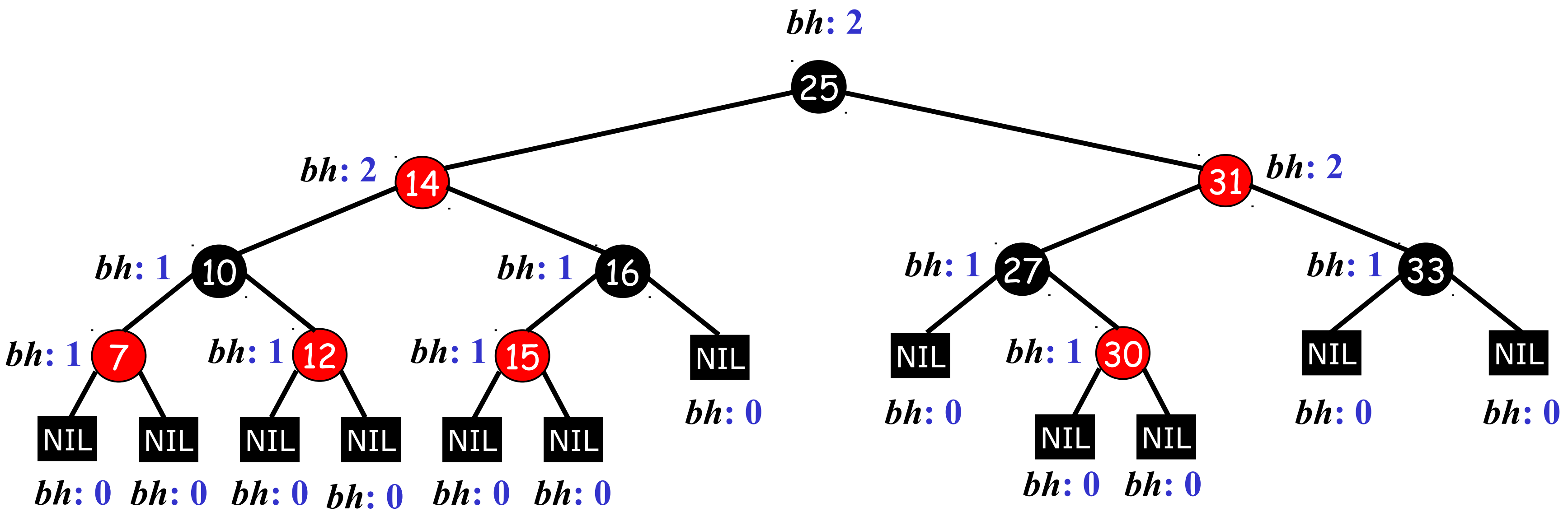
- A red-black tree with n internal nodes has height at most $2\log(n+1)$
  - Consider there are some red and black nodes in RB-Tree
  - Let h be the height of the RB-Tree
  - Let bh be the black height of the RB-Tree
  - This means at least half of the nodes in any path from root, excluding root, are black
  - Therefore bh is at least $h/2$
  - Therefore $n \geq 2^{h/2} -1$
    - Where n is the number of internal nodes (i.e. key bearing nodes)
  - Thus, $h \leq 2\log(n+1)$
- Height of RB-Tree: $\log_4 n \leq h \leq \log_2 n$

Insertion in a RB-Tree

- Initial step in the insertion is same as in BST
- Differences from BST:
  - The left and right children of inserted node are linked to NIL external node
  - The inserted node is colored RED
- The only red-black property violated during the insertion is the property 4
  - If a node is red, then its parent is black
- To restore the red-black property, move the violations up the tree by re-coloring until it can be fixed with rotations and re-coloring

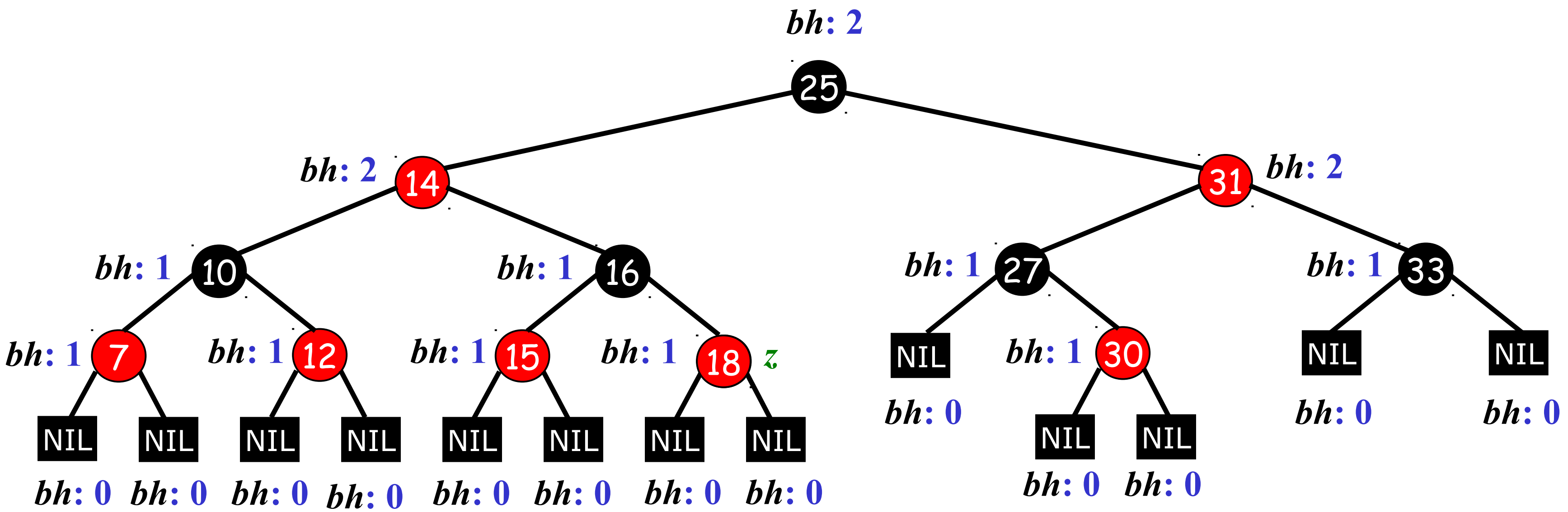
Insertion: Example

- Insert: 18



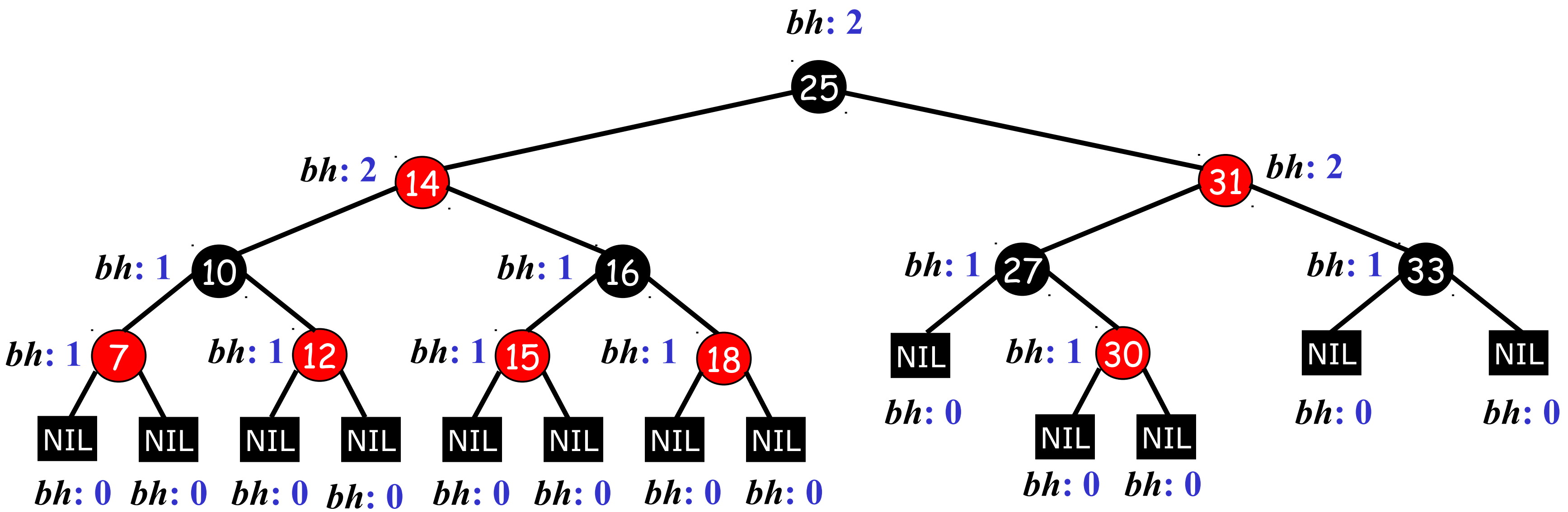
Insertion: Example

- Insert: 18



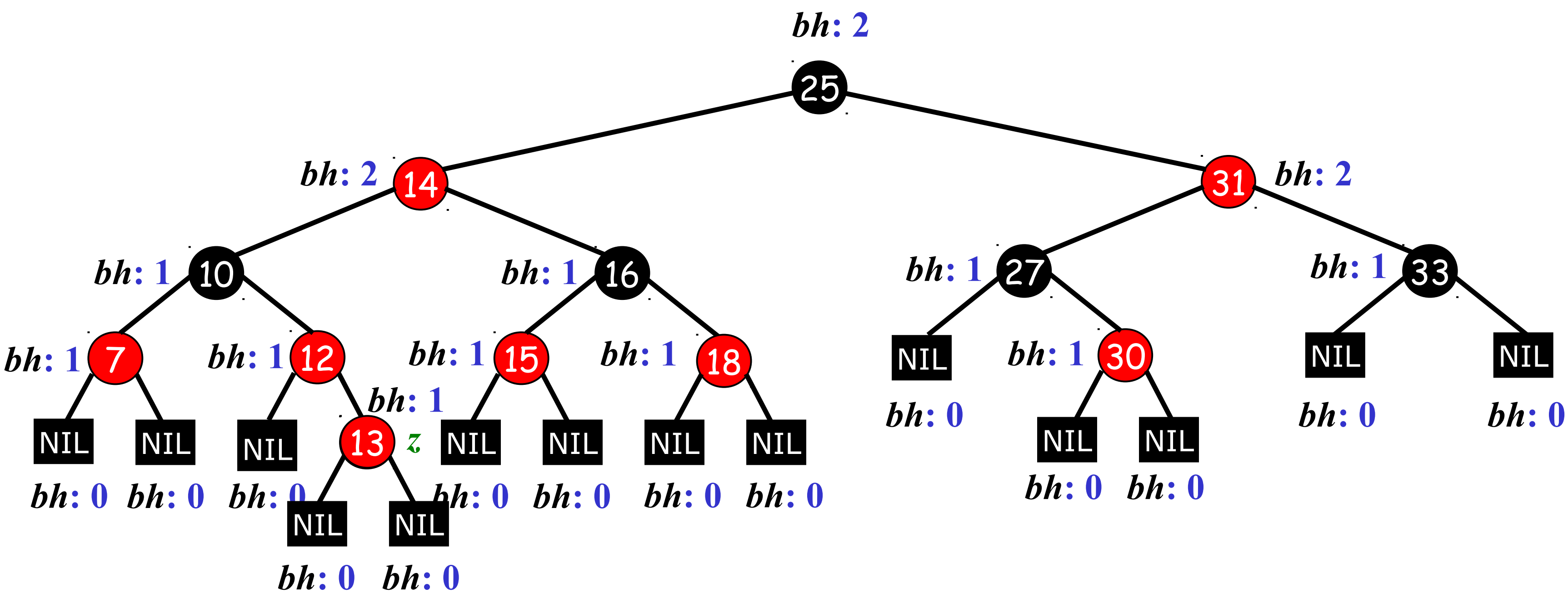
Insertion: Example

- Insert: 13



Insertion: Example

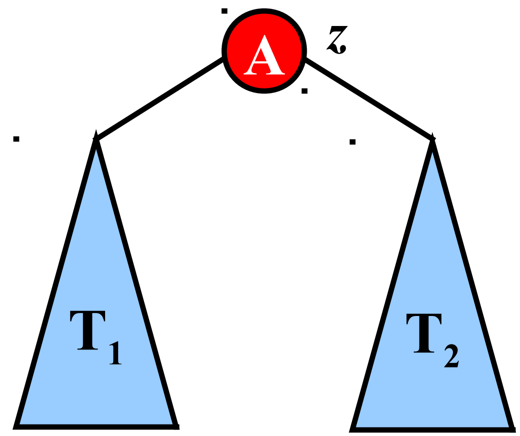
- Insert: 13



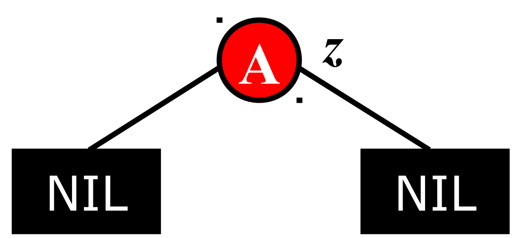
- Property 4 of red-black property is violated

Different Cases to Restore RB-Property after Insertion

- There are 6 cases:
- Let z be the node inserted
- Let the subtree at z be denoted as follows:

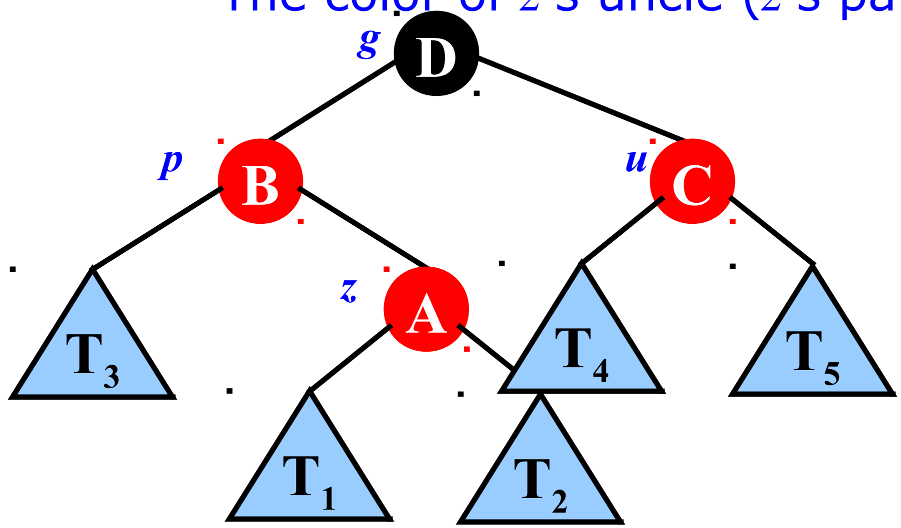


- Initially it will be as follows



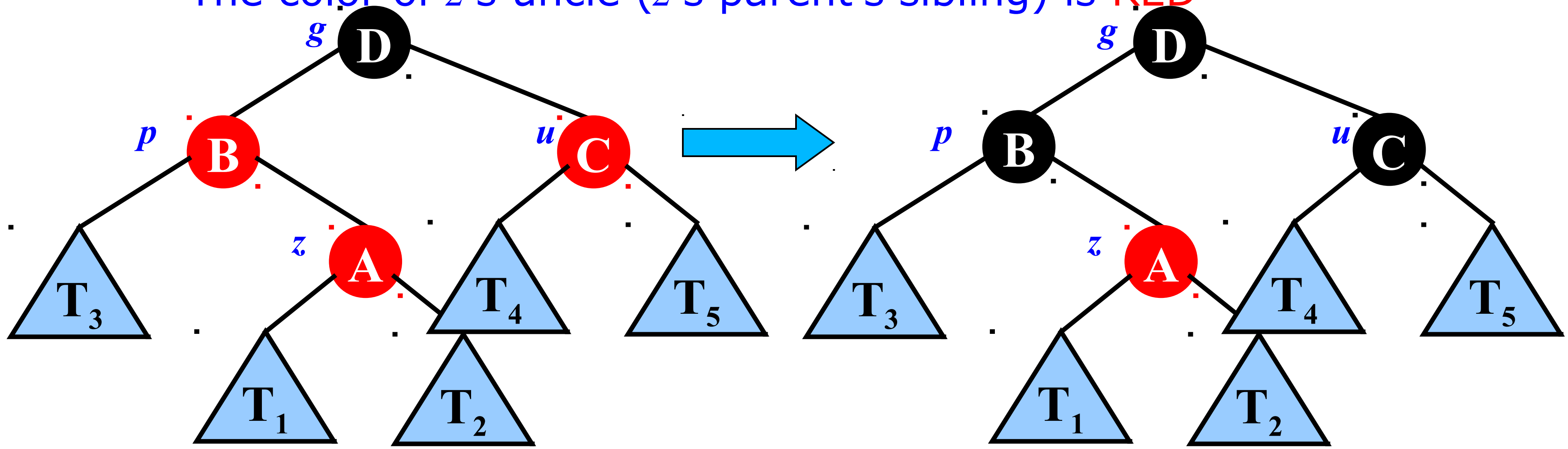
Different Cases to Restore RB-Property after Insertion

- **CASE 1**:
  - The z is left or right child of its parent
  - The parent of z is left child of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is RED



Different Cases to Restore RB-Property after Insertion

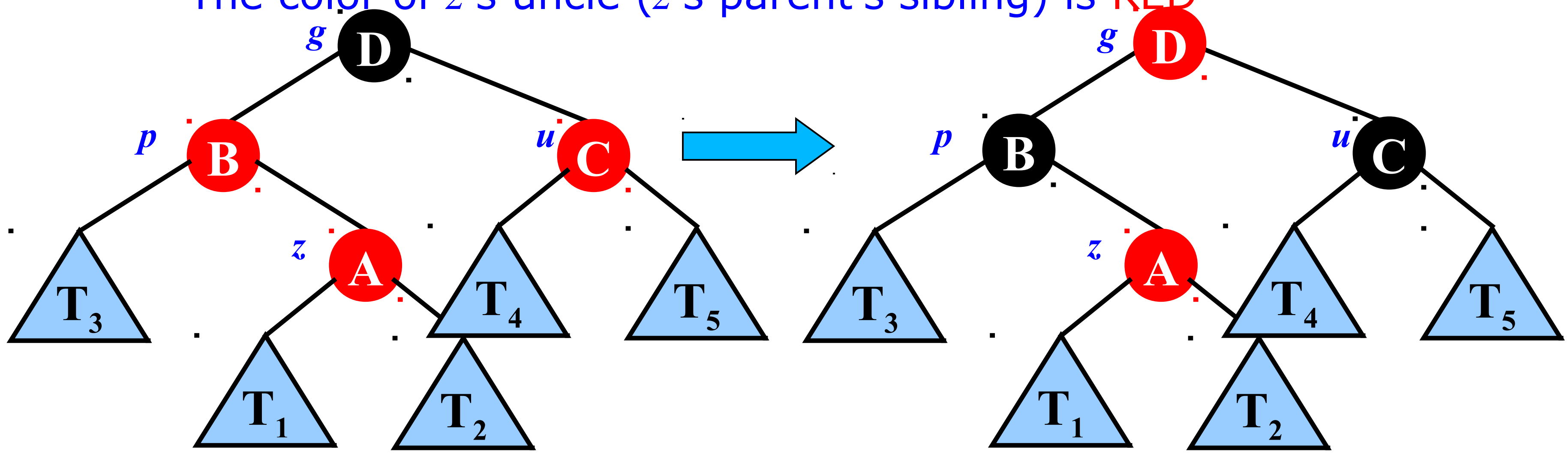
- **CASE 1**:
  - The z is left or right child of its parent
  - The parent of z is left child of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is RED



- Recolor the parent (p) and uncle (u) of z to BLACK

Different Cases to Restore RB-Property after Insertion

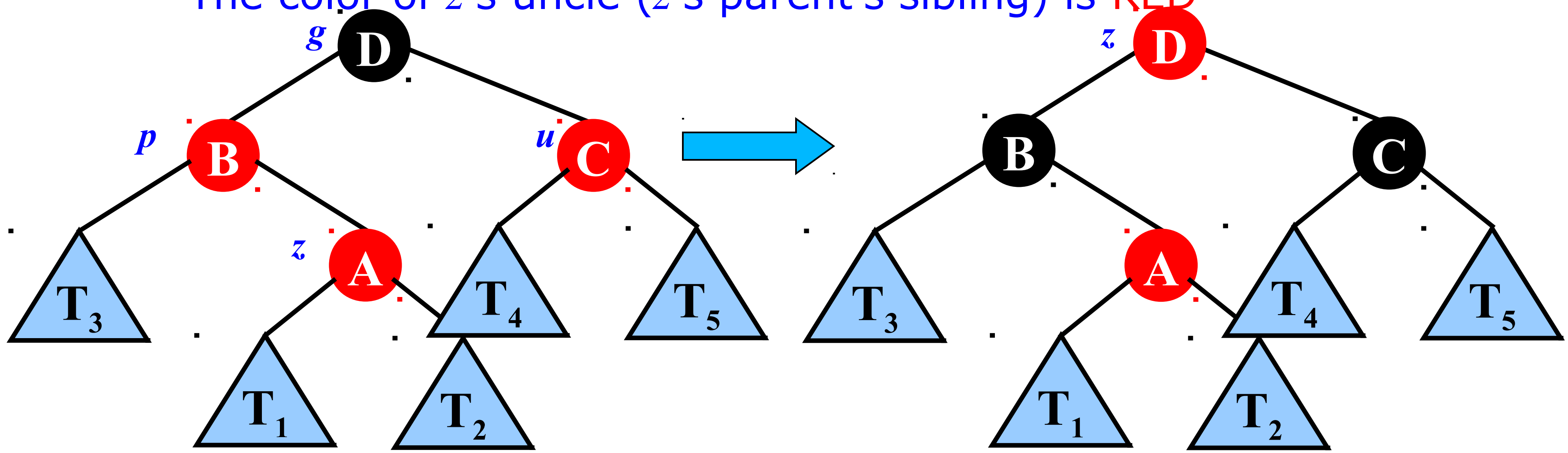
- **CASE 1**:
  - The z is **left or right** child of its parent
  - The parent of z is **left child** of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is RED



- Recolor the parent (p) and uncle (u) of z to BLACK
- Recolor the grand parent (g) of z to RED

Different Cases to Restore RB-Property after Insertion

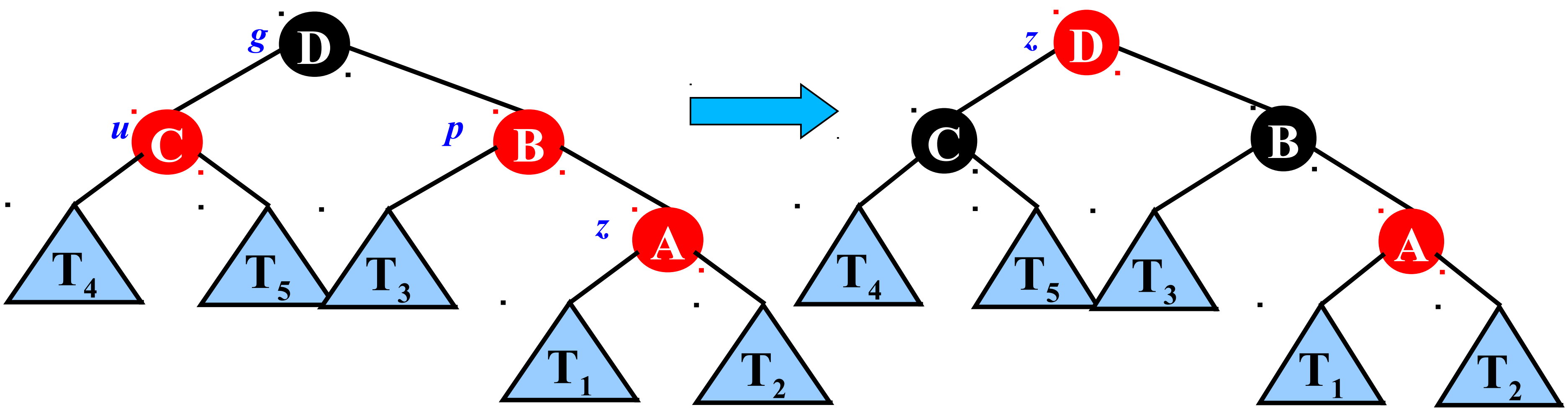
- **CASE 1**:
  - The z is left or right child of its parent
  - The parent of z is left child of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is RED



- Recolor the parent (p) and uncle (u) of z to BLACK
- Recolor the grand parent (g) of z to RED
- Make grand parent (g) of z as new z

Different Cases to Restore RB-Property after Insertion

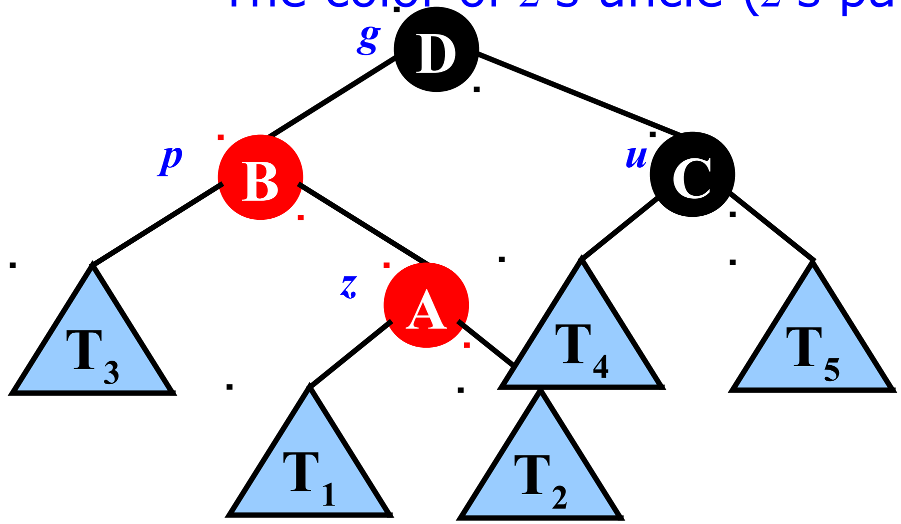
- **CASE 4**: Mirror of CASE 1
  - The z is **left or right** child of its parent
  - The parent of z is **right child** of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is RED



- Recolor the parent (p) and uncle (u) of z to BLACK
- Recolor the grand parent (g) of z to RED
- Make grand parent (g) of z as new z

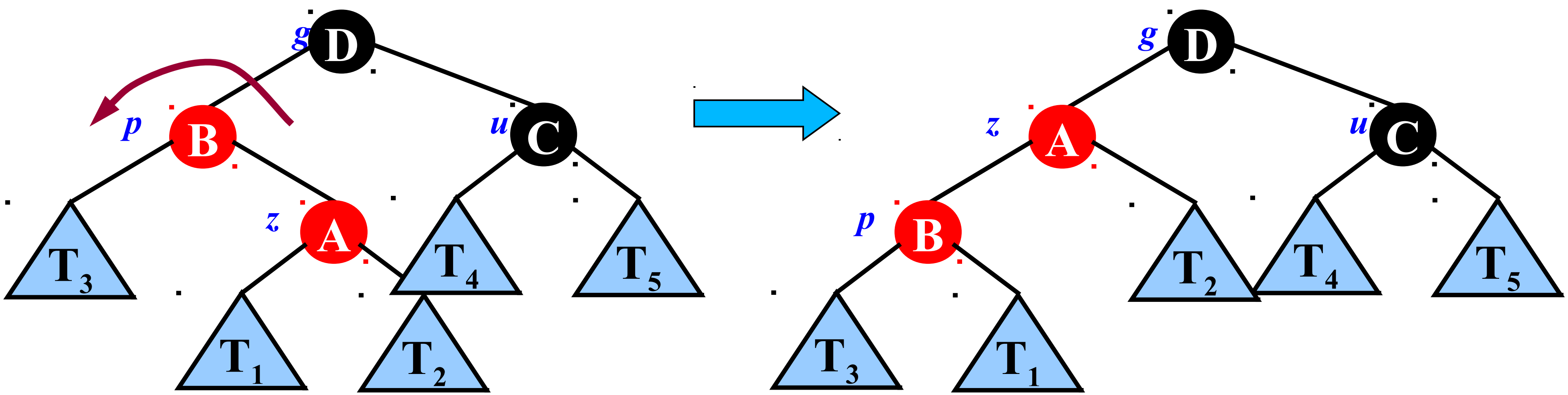
Different Cases to Restore RB-Property after Insertion

- **CASE 2**:
  - The z is **right child** of its parent
  - The parent of z is **left child** of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



Different Cases to Restore RB-Property after Insertion

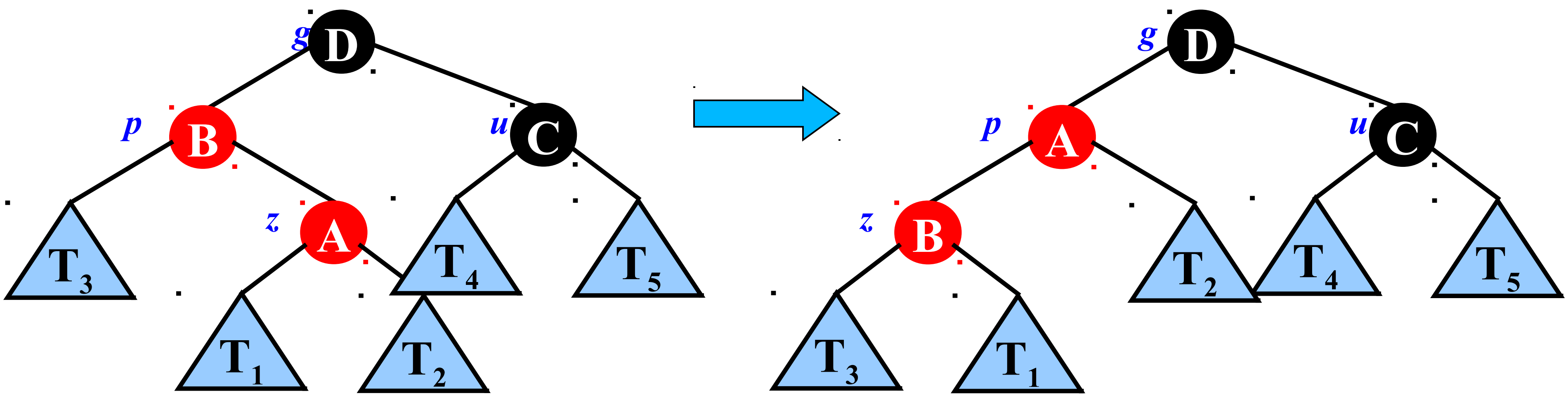
- **CASE 2**:
  - The z is right child of its parent
  - The parent of z is left child of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



- Do left-rotation (LL) at parent (p) of z

Different Cases to Restore RB-Property after Insertion

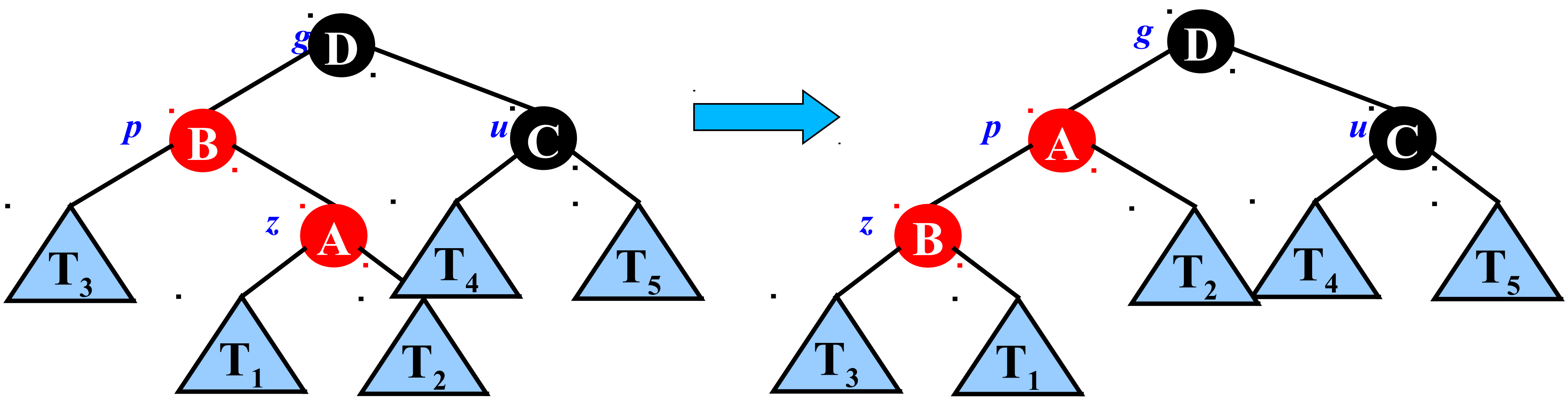
- **CASE 2**:
  - The z is right child of its parent
  - The parent of z is left child of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



- Do left-rotation (LL) at parent (p) of z
- Make earlier parent (p) of z as new z

Different Cases to Restore RB-Property after Insertion

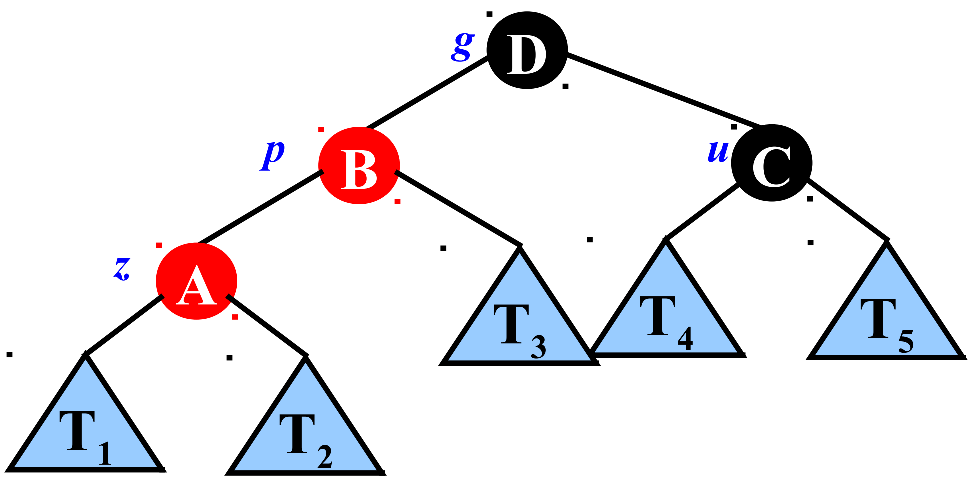
- **CASE 2**:
  - The z is right child of its parent
  - The parent of z is left child of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



- Do left-rotation (LL) at parent (p) of z
- Make earlier parent (p) of z as new z
- **This leads to CASE 3**

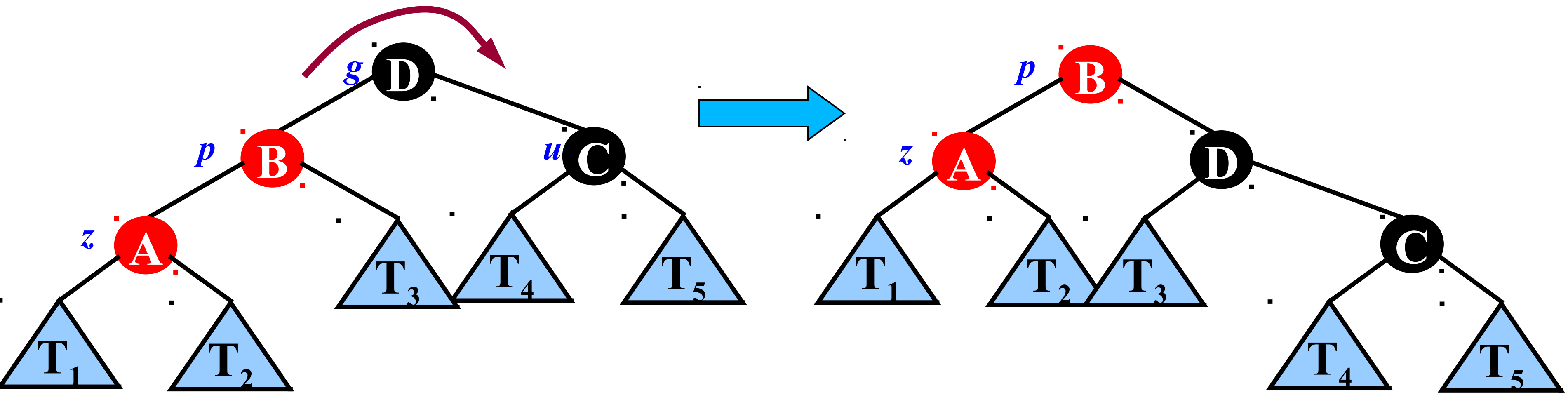
Different Cases to Restore RB-Property after Insertion

- **CASE 3**:
  - The z is left child of its parent
  - The parent of z is left child of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



Different Cases to Restore RB-Property after Insertion

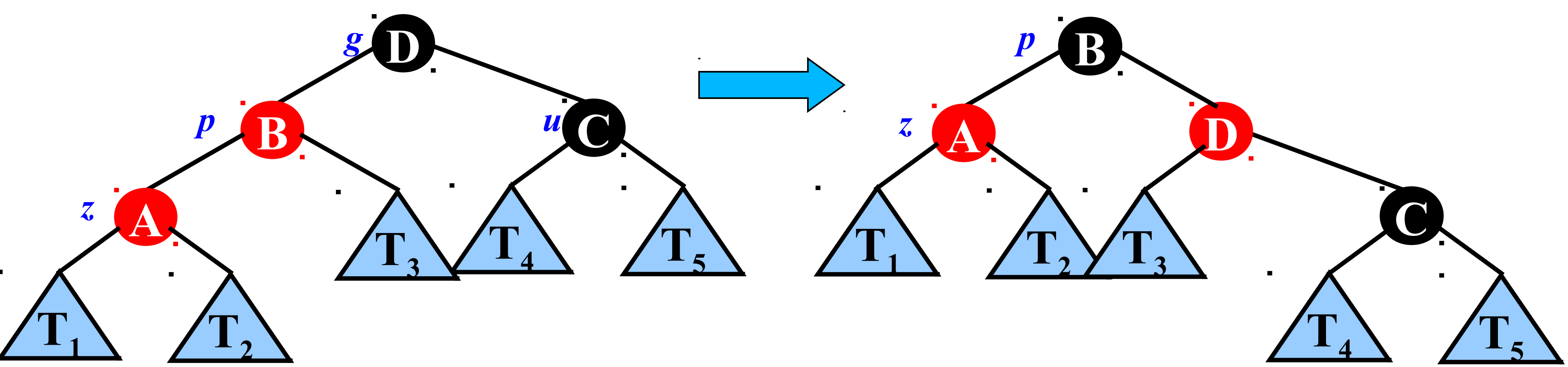
- **CASE 3**:
  - The z is left child of its parent
  - The parent of z is left child of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



- Do right-rotation (RR) at grand parent (p) of z

Different Cases to Restore RB-Property after Insertion

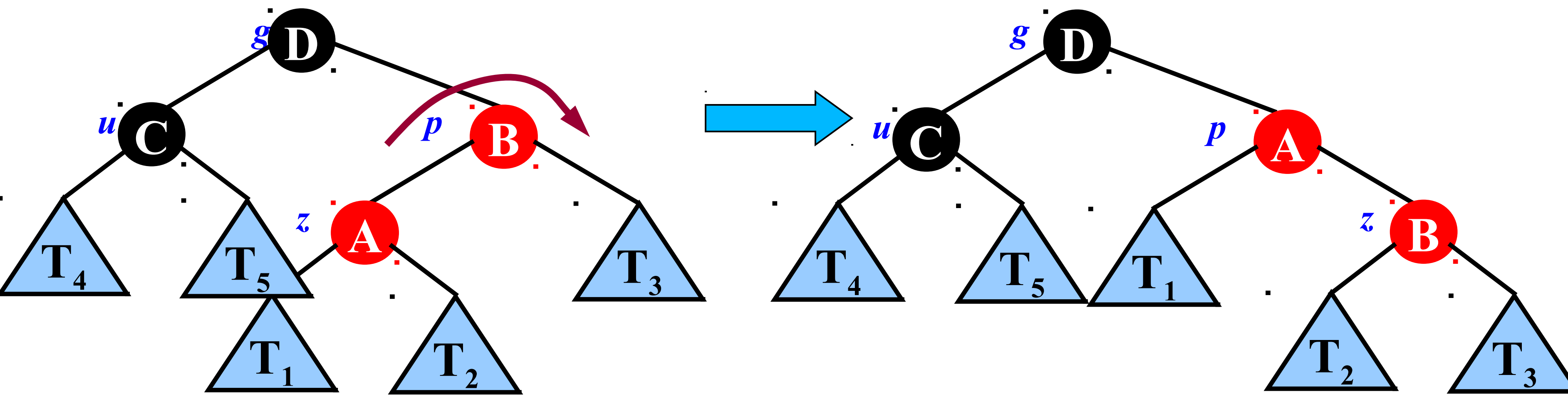
- **CASE 3**:
  - The z is **left child** of its parent
  - The parent of z is **left child** of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



- Do right-rotation (RR) at grand parent (g) of z
- Recolor the parent (p) of z to BLACK
- Recolor the children of parent (p) of z to RED

Different Cases to Restore RB-Property after Insertion

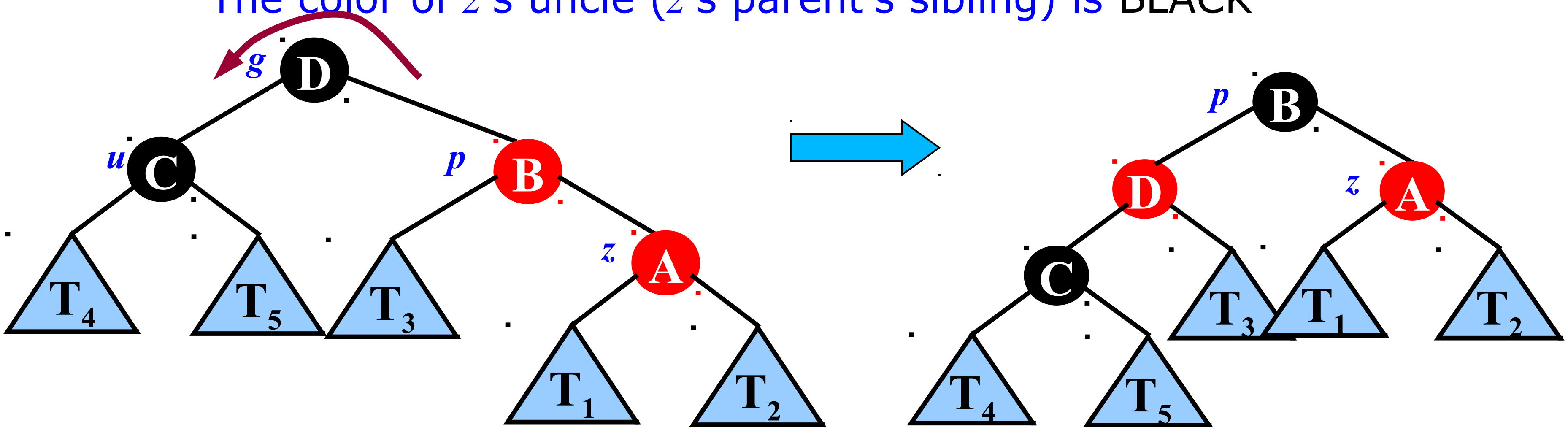
- **CASE 5**: Mirror of CASE 2
  - The z is **left child** of its parent
  - The parent of z is **right child** of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



- Do right-rotation (RR) at parent (p) of z
- Make earlier parent (p) of z as new z
- **This leads to CASE 6**

Different Cases to Restore RB-Property after Insertion

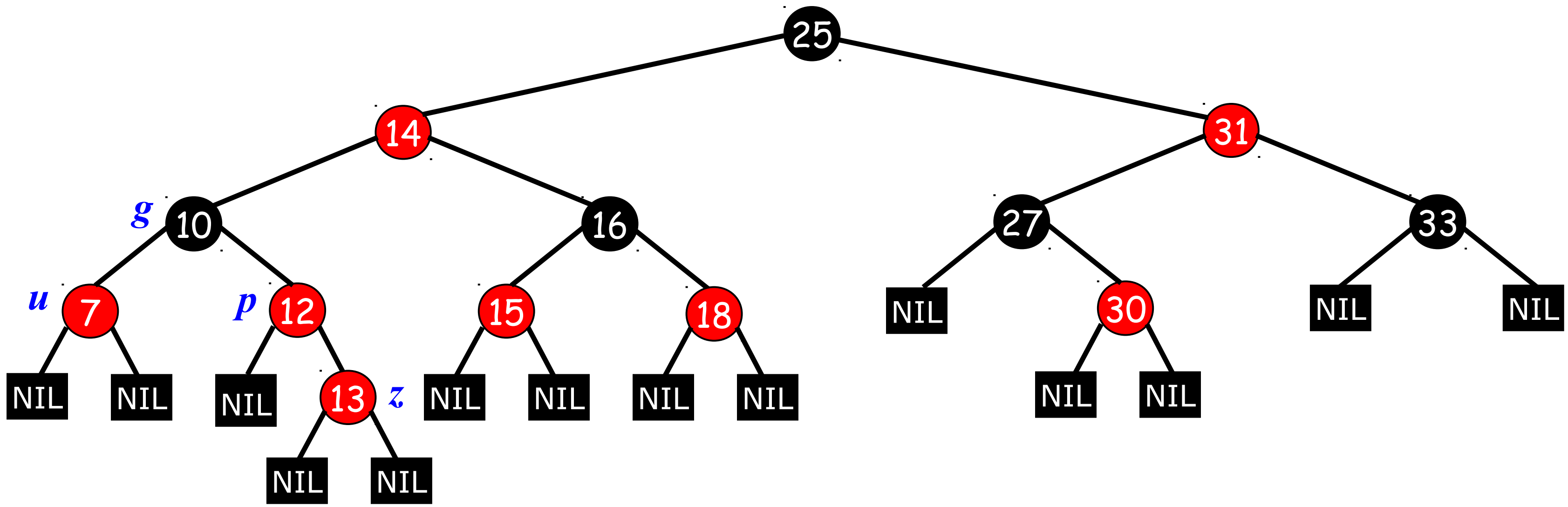
- **CASE 6**: Mirror of CASE 3
  - The z is **right child** of its parent
  - The parent of z is **right child** of grand parent of z
  - The color of parent of z is RED
  - The color of z's uncle (z's parent's sibling) is BLACK



- Do left-rotation (LL) at grand parent (g) of z
- Recolor the parent (p) of z to BLACK
- Recolor the children of parent (p) of z to RED

Insertion: Example

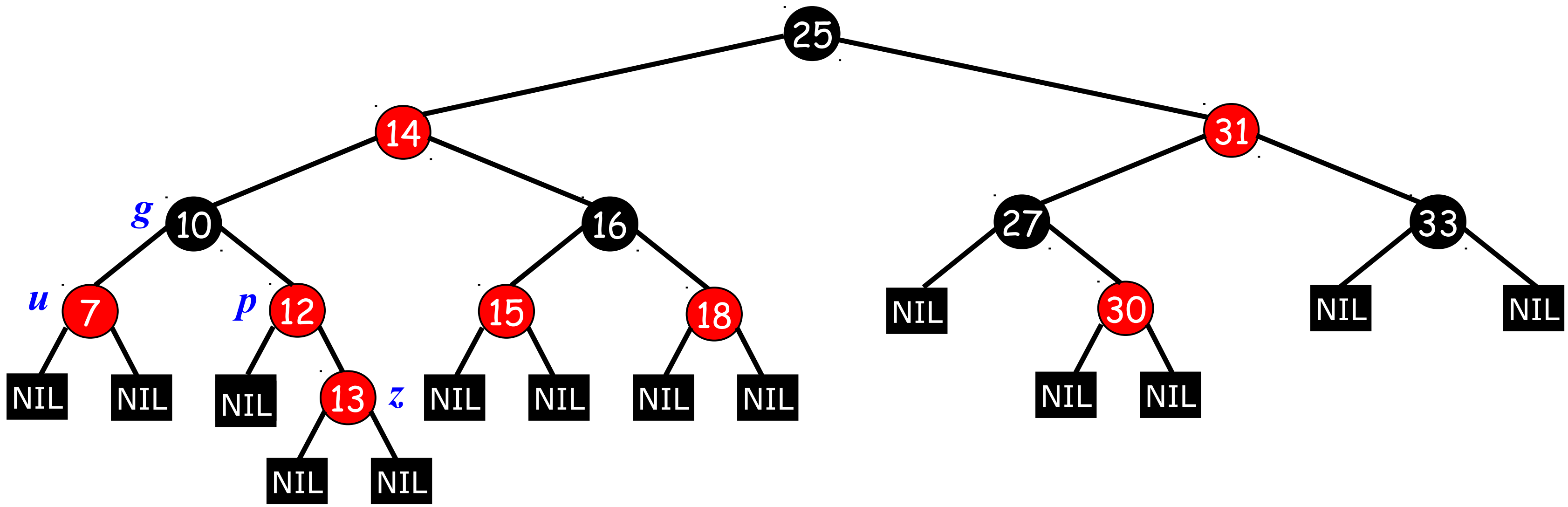
- Insert: 13

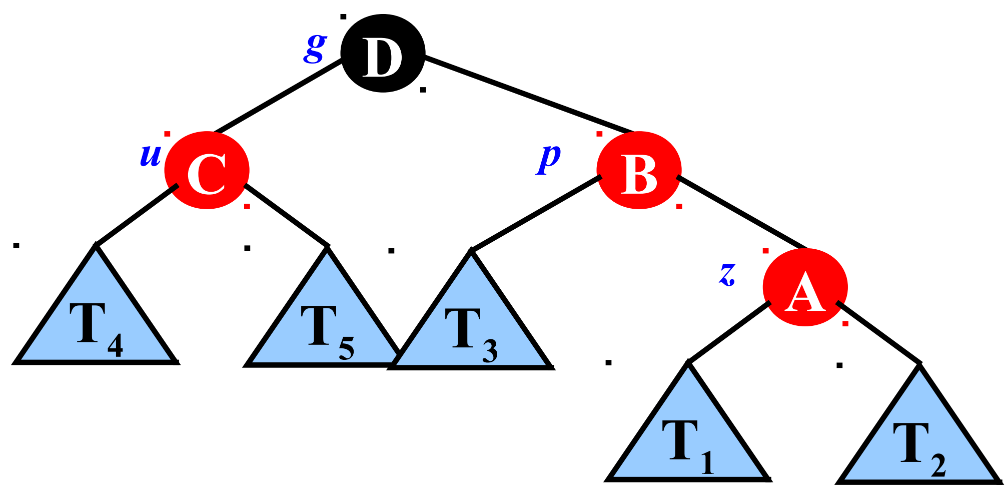


- Property 4 of red-black property is violated
- CASE 4:

Insertion: Example

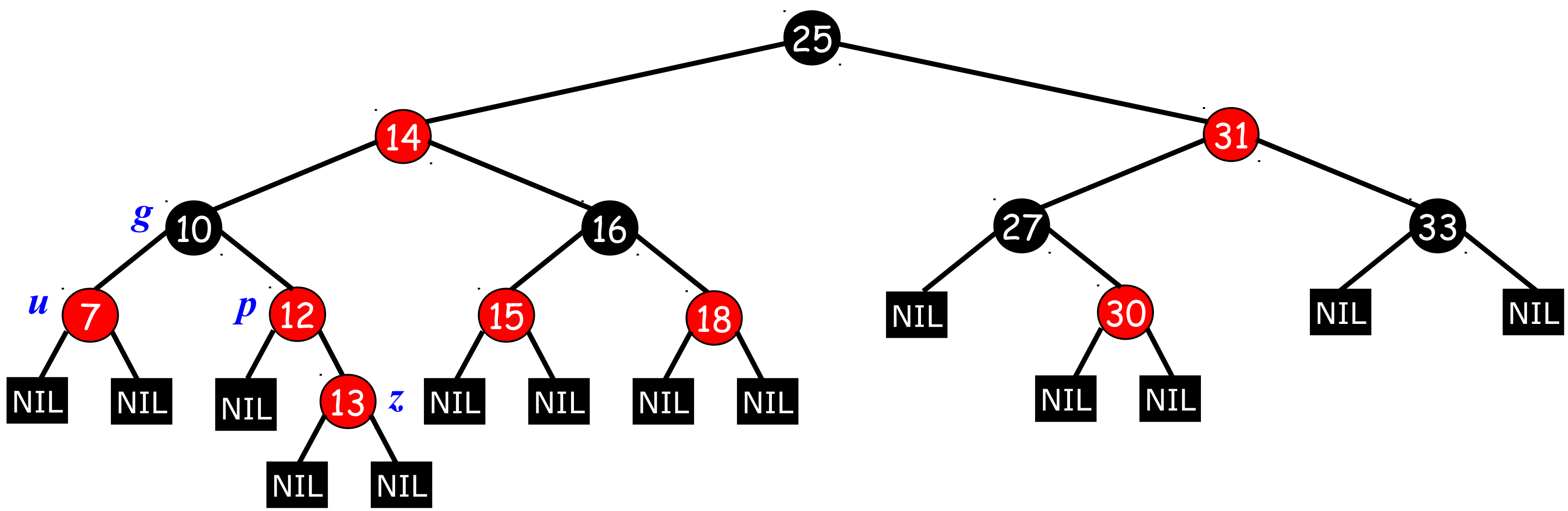
- Insert: 13





Insertion: Example

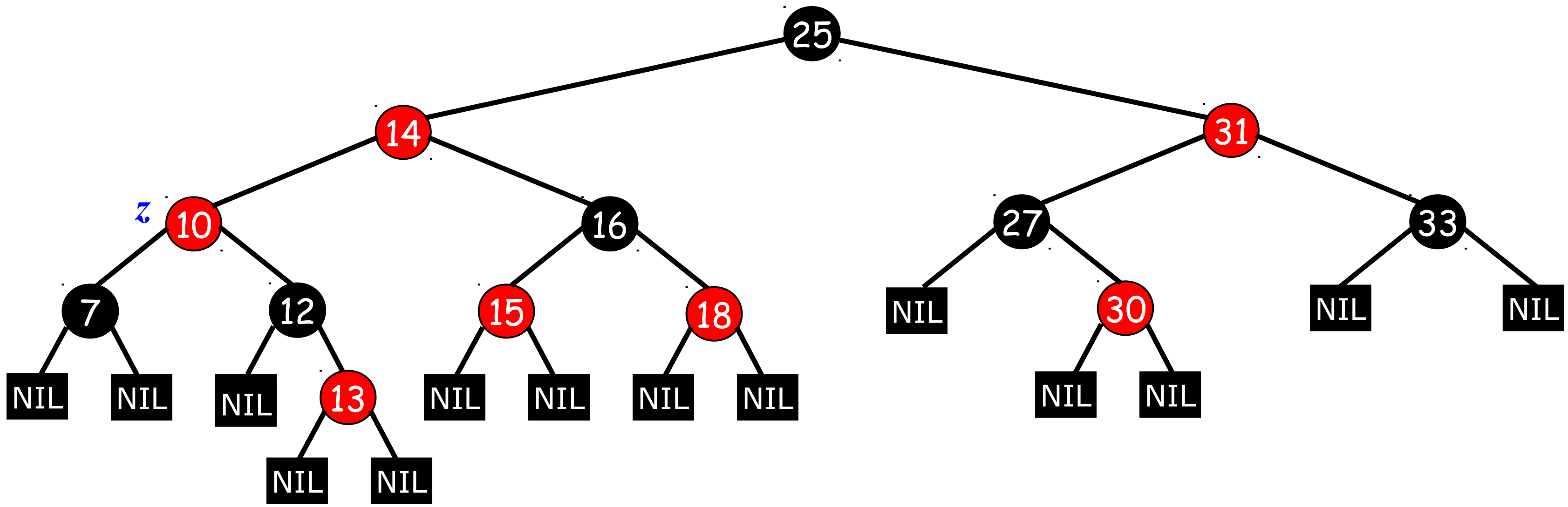
- Insert: 13



- Property 4 of red-black property is violated
- CASE 4:
  - Recolor the parent (p) and uncle (u) of z to BLACK
  - Recolor the grand parent (g) of z to RED
  - Make grand parent (g) of z as new z

Insertion: Example

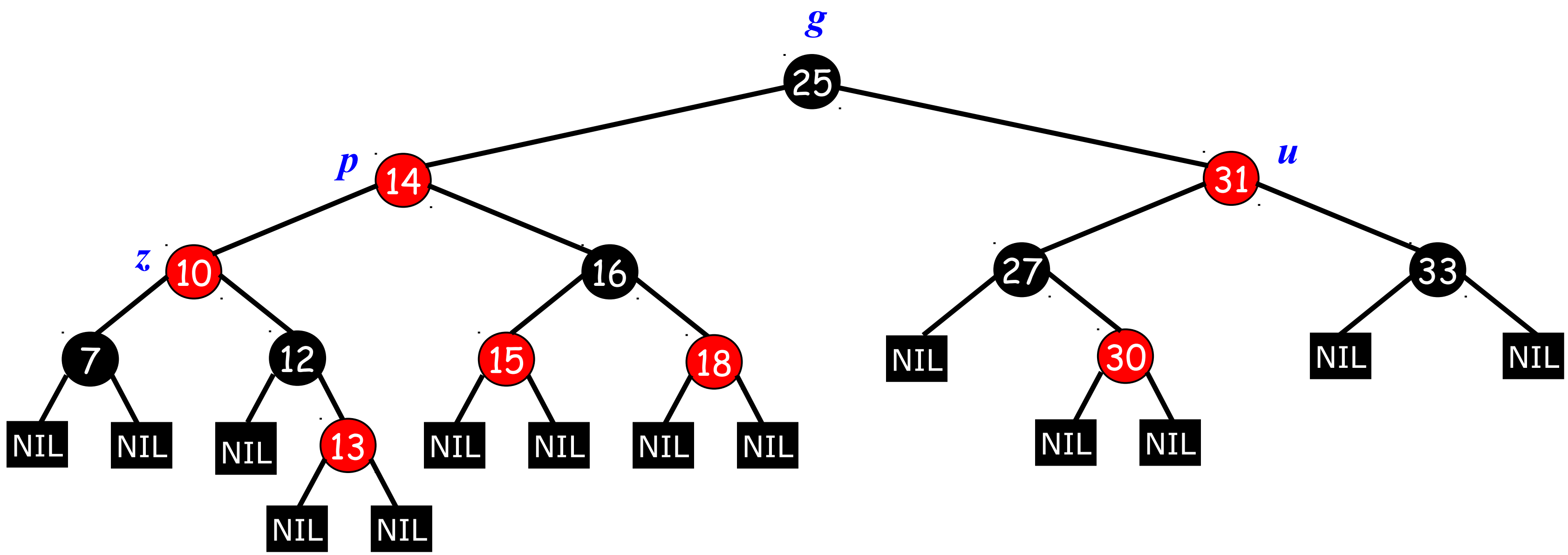
- Insert: 13



- CASE 4:
  - Recolor the parent (p) and uncle (u) of z to BLACK
  - Recolor the grand parent (g) of z to RED
  - Make grand parent (g) of z as new z

Insertion: Example

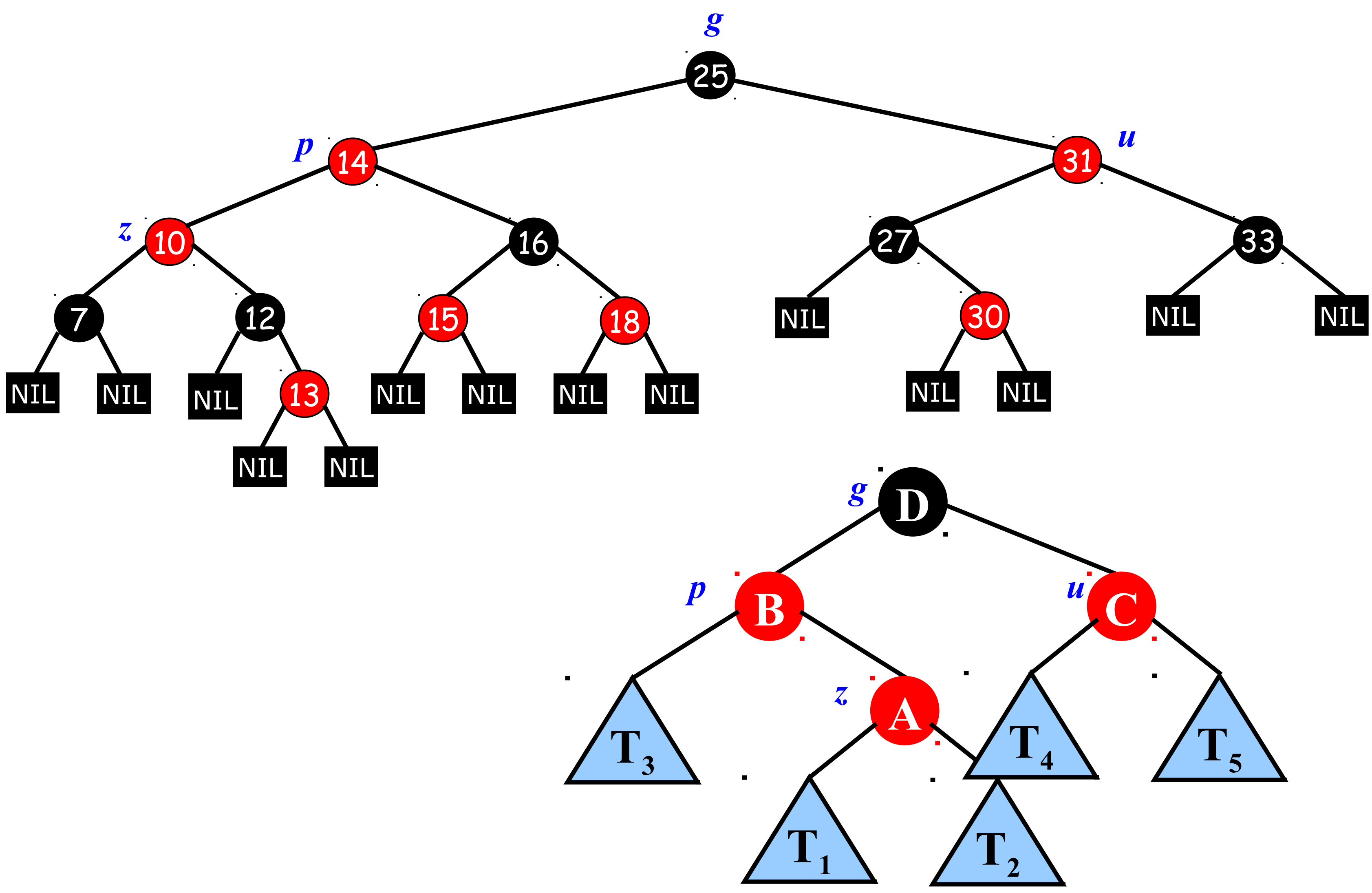
- Insert: 13



- Property 4 of red-black property is violated
- CASE 1:

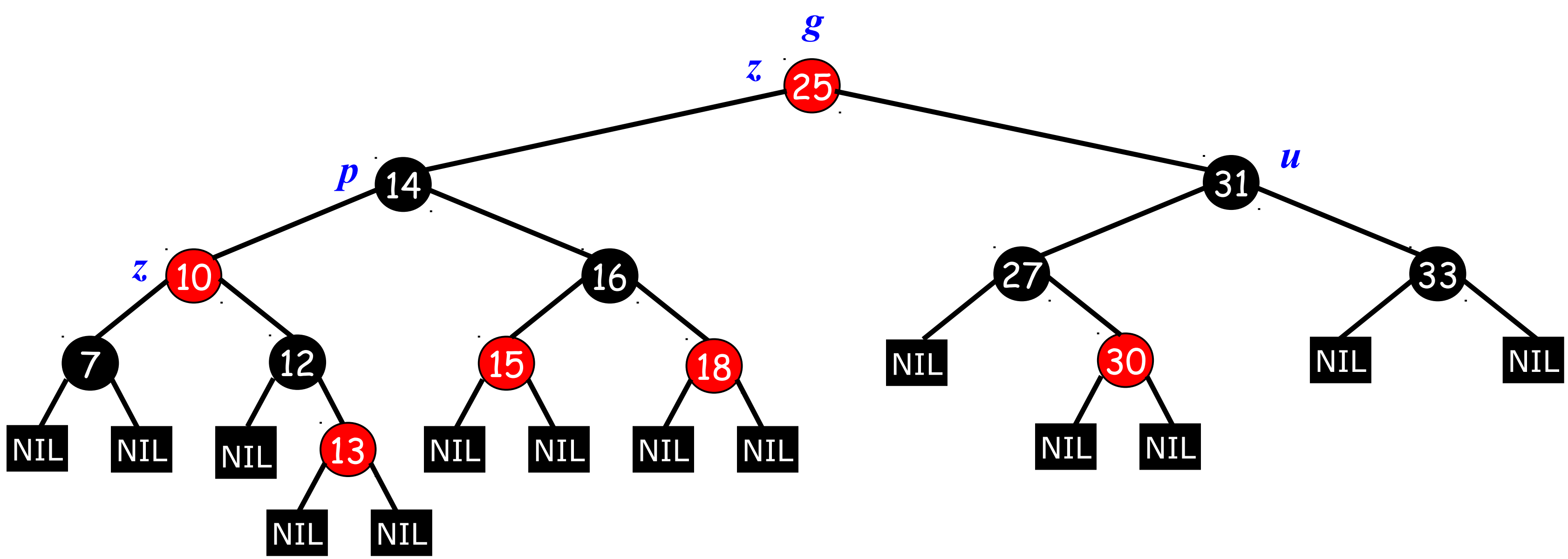
Insertion: Example

- Insert: 13



Insertion: Example

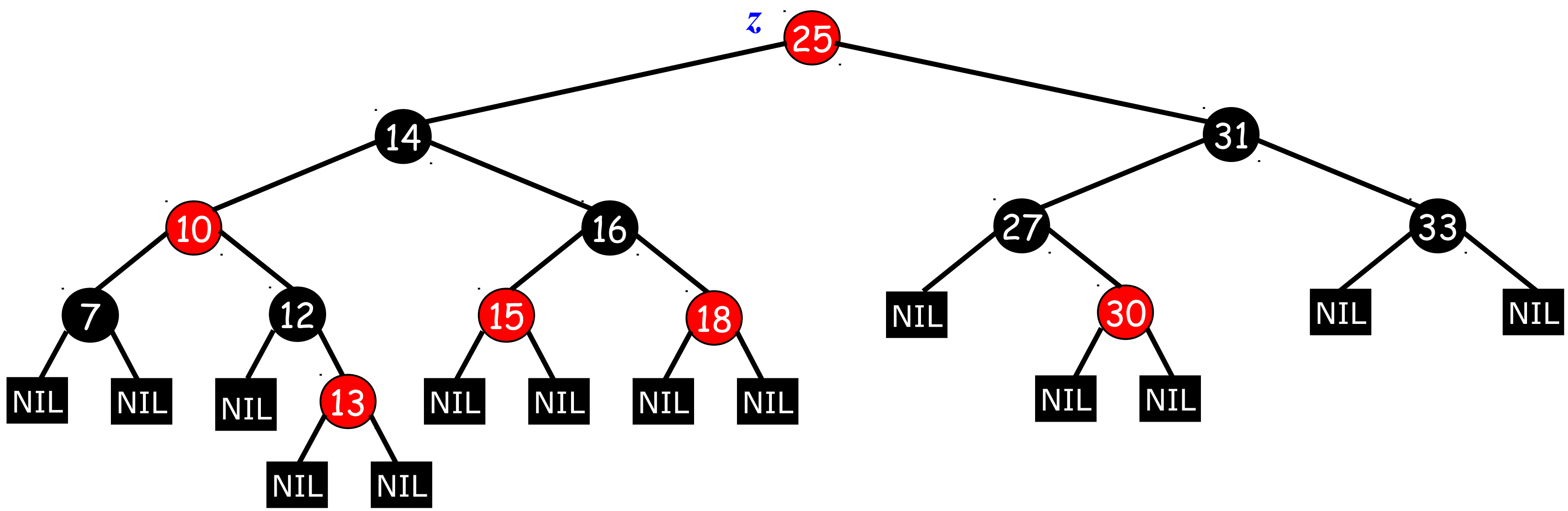
- Insert: 13



- CASE 1:
  - Recolor the parent (p) and uncle (u) of z to BLACK
  - Recolor the grand parent (g) of z to RED
  - Make grand parent (g) of z as new z

Insertion: Example

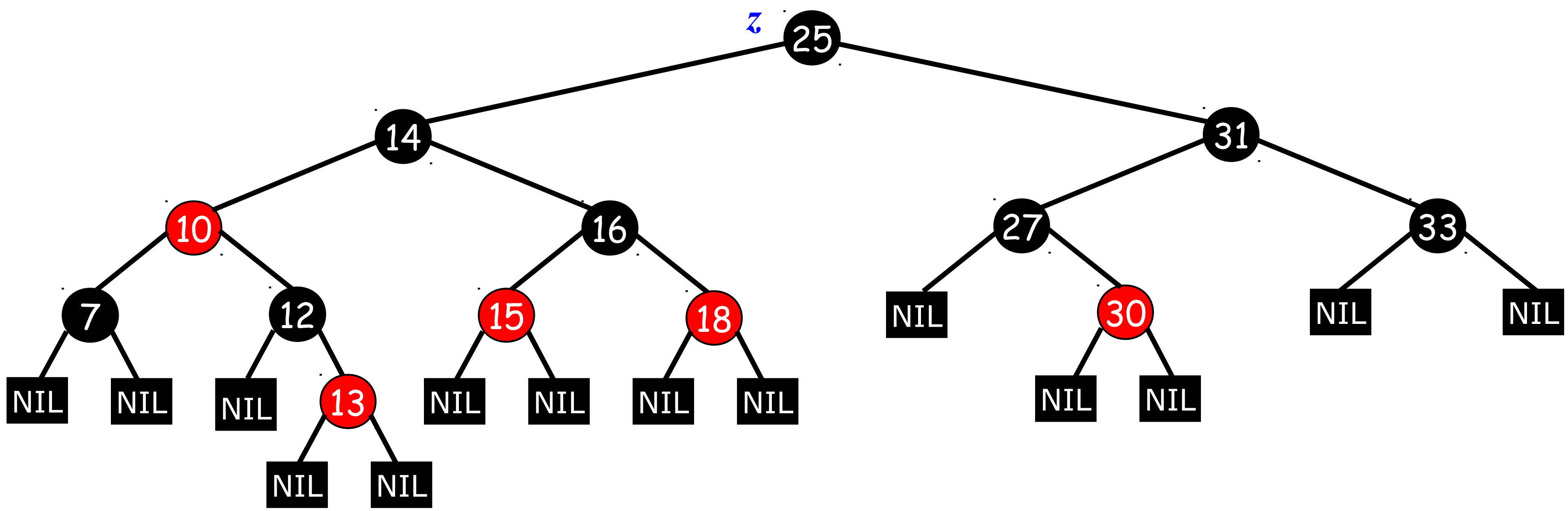
- Insert: 13



- z has reached the root
- **Property 2 violation**: Root is always black
- Restoring property 2: **Recolor root to BLACK**

Insertion: Example

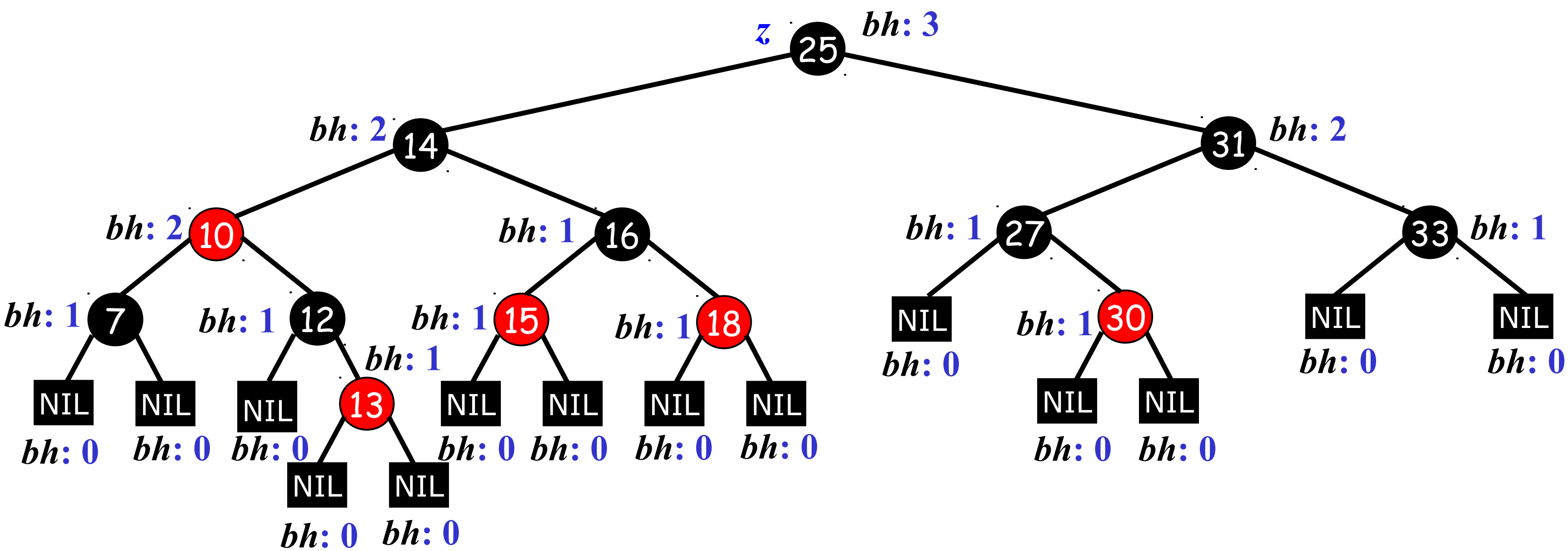
- Insert: 13



- z has reached the root
- **Property 2 violation**: Root is always black
- Restoring property 2: **Recolor root to BLACK**

Insertion: Example

- Balanced RB-Tree after insertion of 13



- Black-height of RB-Tree: 3

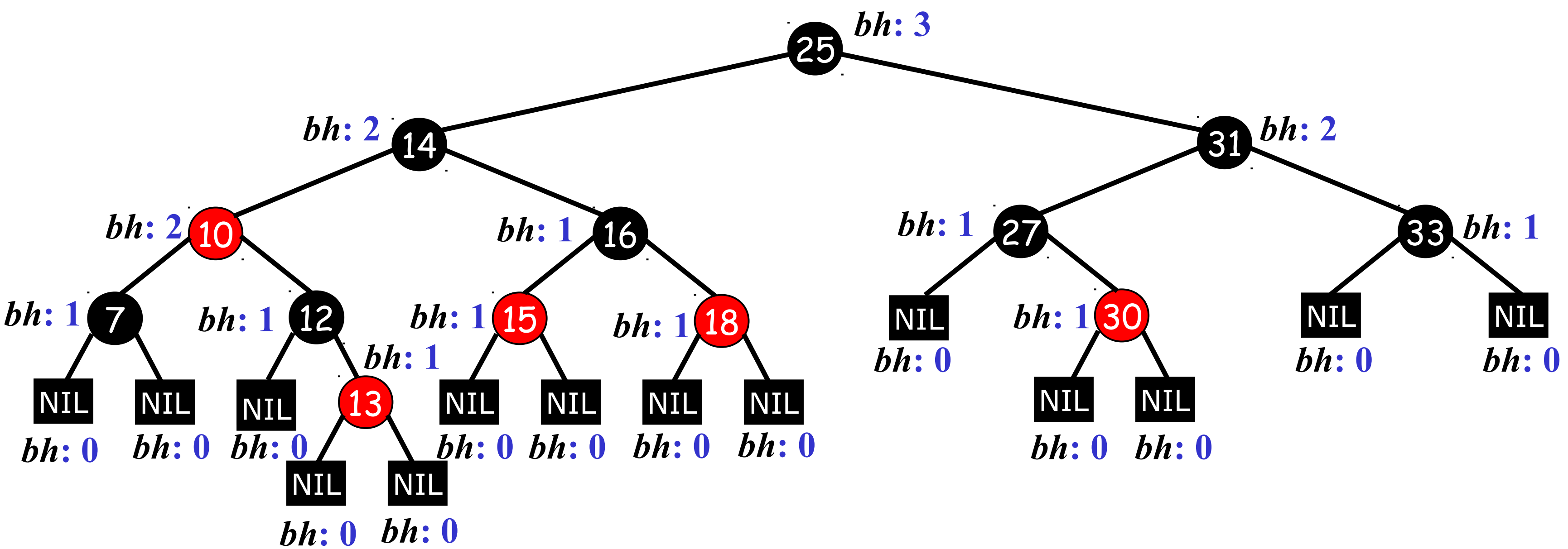
Pseudo-code for Insert and Restoring RB-property

```
RB-INSERT(z)
// z is the node to be inserted
    BST-INSERT(z)
    z → leftChild = EN-NIL      // EN-NIL: External node NIL
    z → rightChild = EN-NIL
    z → color = RED
    RB-INSERT-FIXUP(z)    // To restore RB-properties
```

```
RB-INSERT-FIXUP(z)
// z is the node inserted
    p = z → parent
    while (p → color == RED)  // Property 4 is being violated
            // Handle CASE 1 to CASE 6. Refer CORMEN Book
    end
    if (p == EN-NIL and z → color == RED)   // z is root
            z → color = BLACK
```

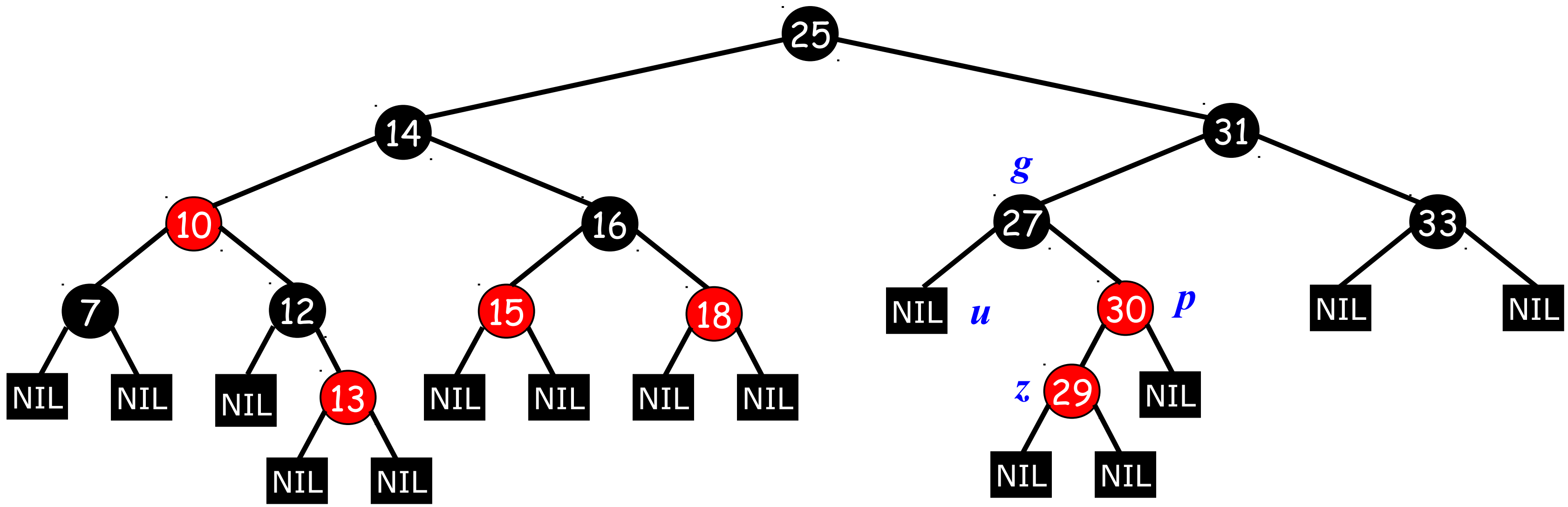
Insertion: Example

- Insert 29



Insertion: Example

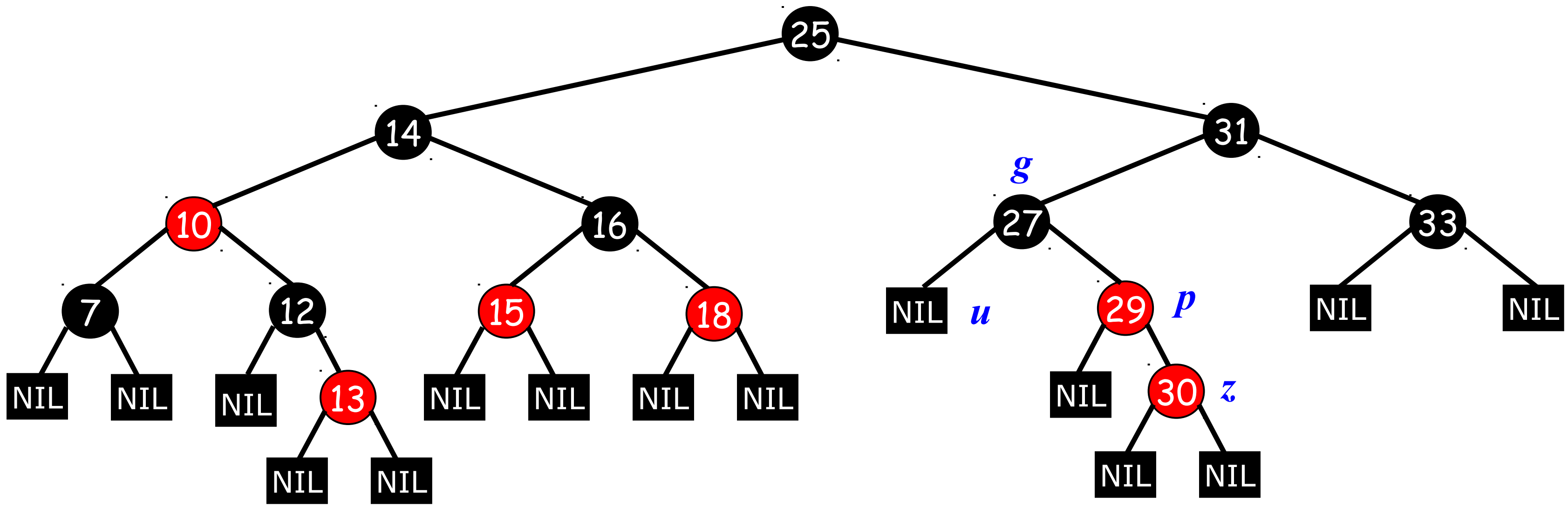
- Insert 29



- Property 4 of red-black property is violated
- CASE 5:
  - Do right-rotation (RR) at parent (p) of z
  - Make earlier parent (p) of z as new z
  - This leads to CASE 6

Insertion: Example

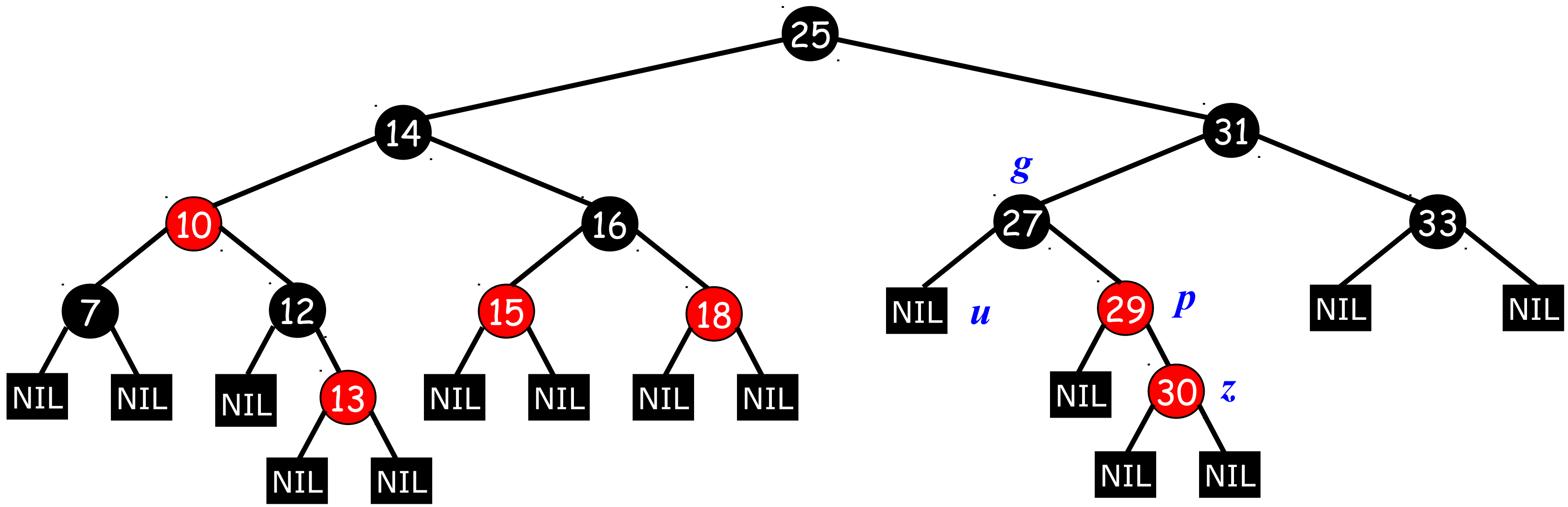
- Insert 29



- Property 4 of red-black property is violated
- CASE 5:
  - Do right-rotation (RR) at parent (p) of z
  - Make earlier parent (p) of z as new z
  - This leads to CASE 6

Insertion: Example

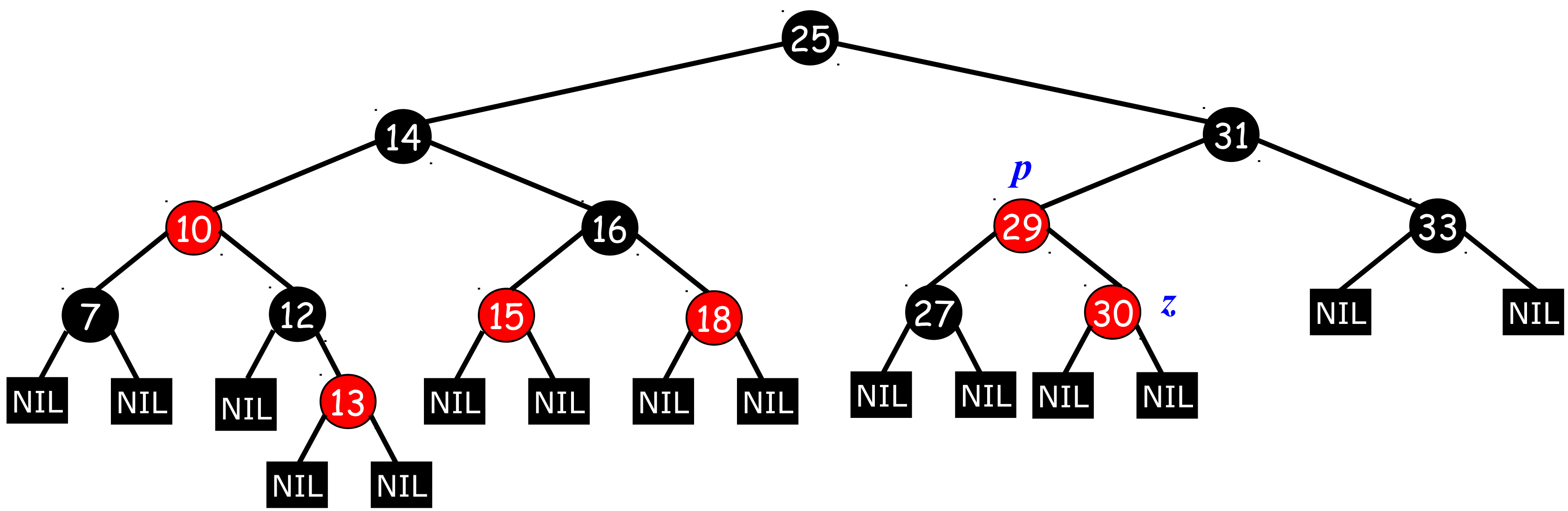
- Insert 29



- Property 4 of red-black property is violated
- CASE 6:
  - Do left-rotation (LL) at grand parent (g) of z
  - Recolor the parent (p) of z to BLACK
  - Recolor the children of parent (p) of z to RED

Insertion: Example

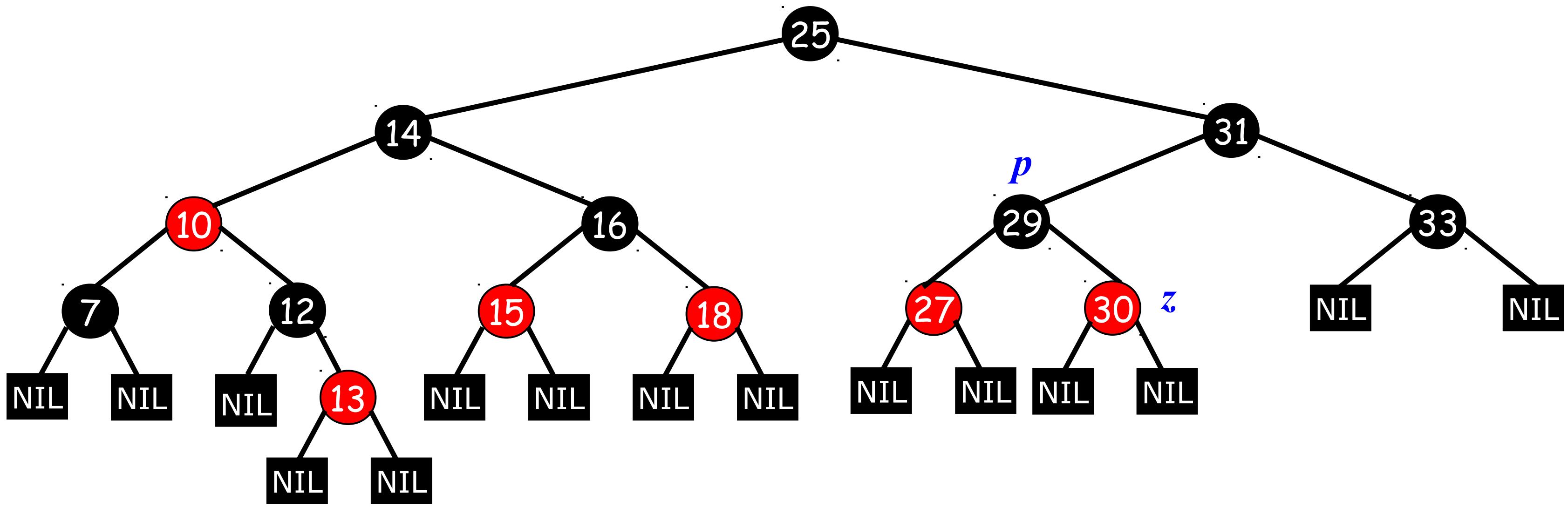
- Insert 29



- Property 4 of red-black property is violated
- CASE 6:
  - Do left-rotation (LL) at grand parent (p) of z
  - Recolor the parent (p) of z to BLACK
  - Recolor the children of parent (p) of z to RED

Insertion: Example

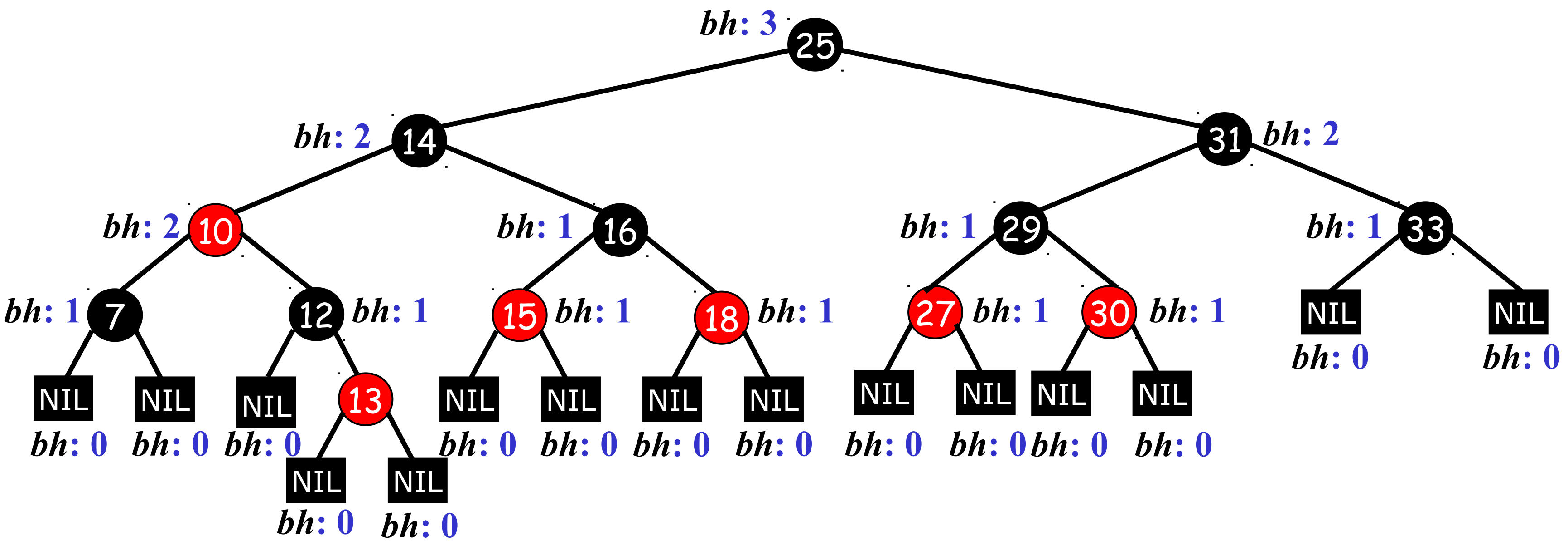
- Insert 29



- Property 4 of red-black property is violated
- CASE 6:
  - Do left-rotation (LL) at grand parent (p) of z
  - Recolor the parent (p) of z to BLACK
  - Recolor the children of parent (p) of z to RED

Insertion: Example

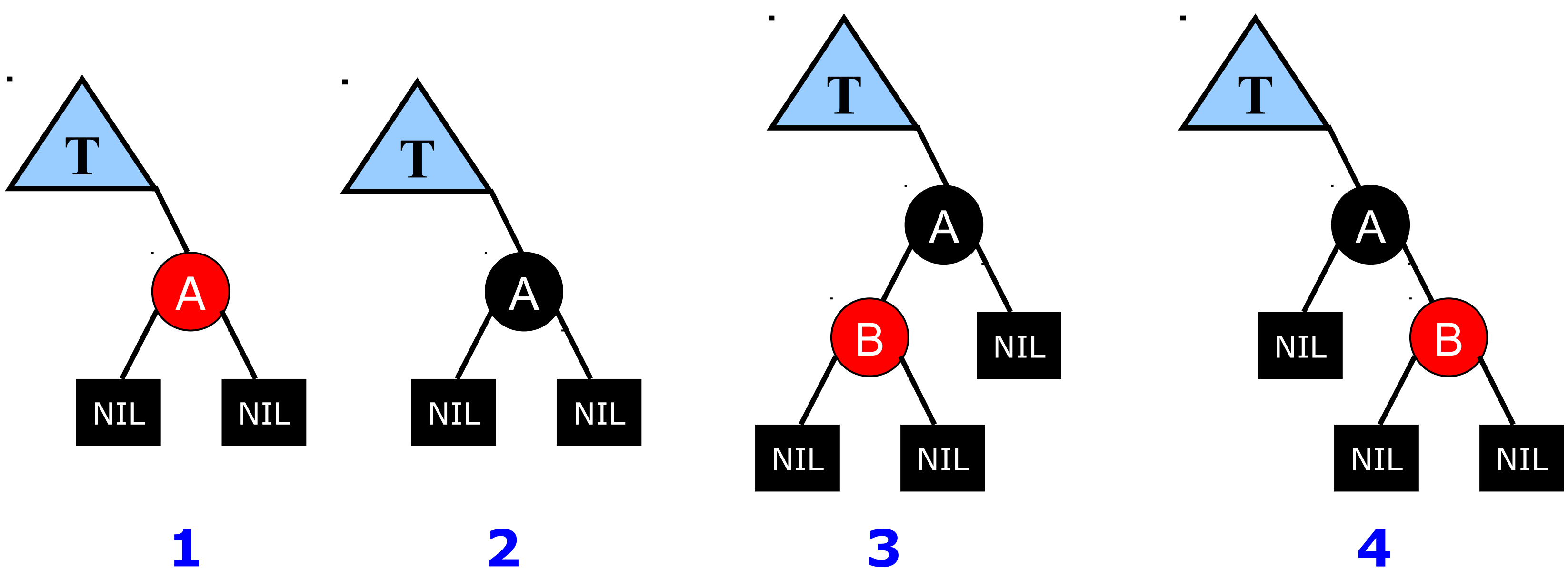
- Balanced RB-Tree after insertion of 29



- Black-height of RB-Tree: 3

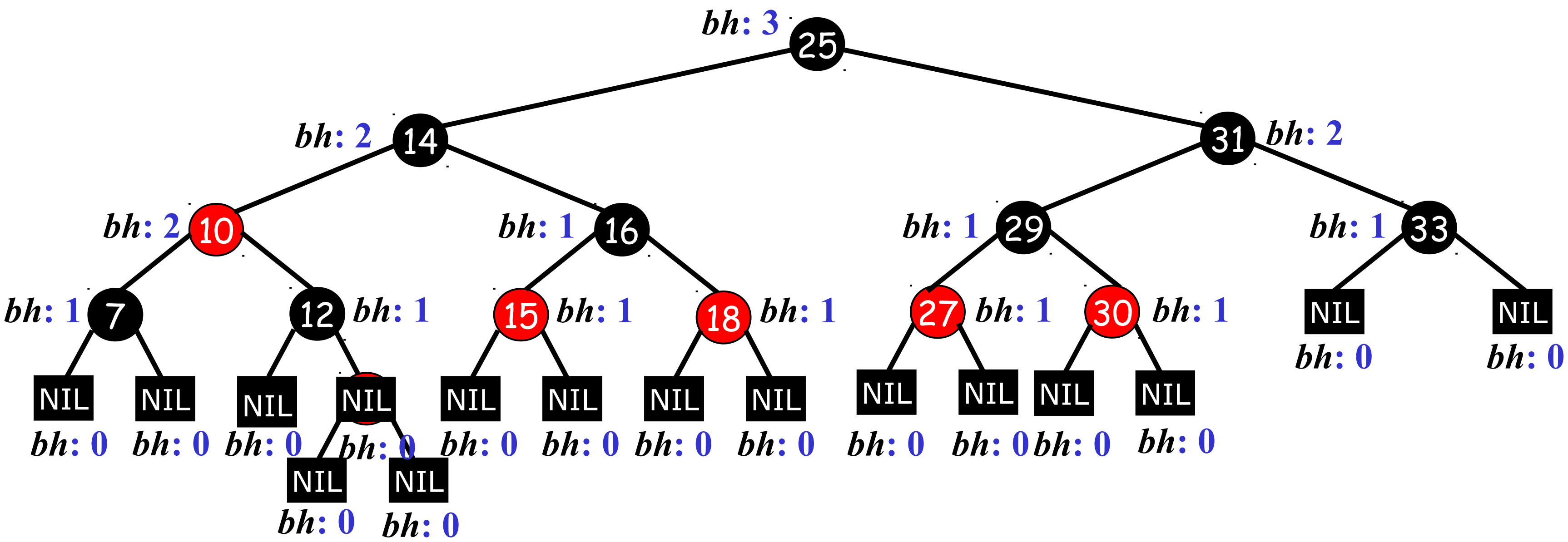
Deletion in a RB-Tree

- Initial step in the deletion is same as in BST
- The node which is deleted is always the parent of some external node
- Following are the only 4 possibilities while deleting a node:
  - While deleting a node A, subtree rooted at A mabe a left or right subtree of bigger tree



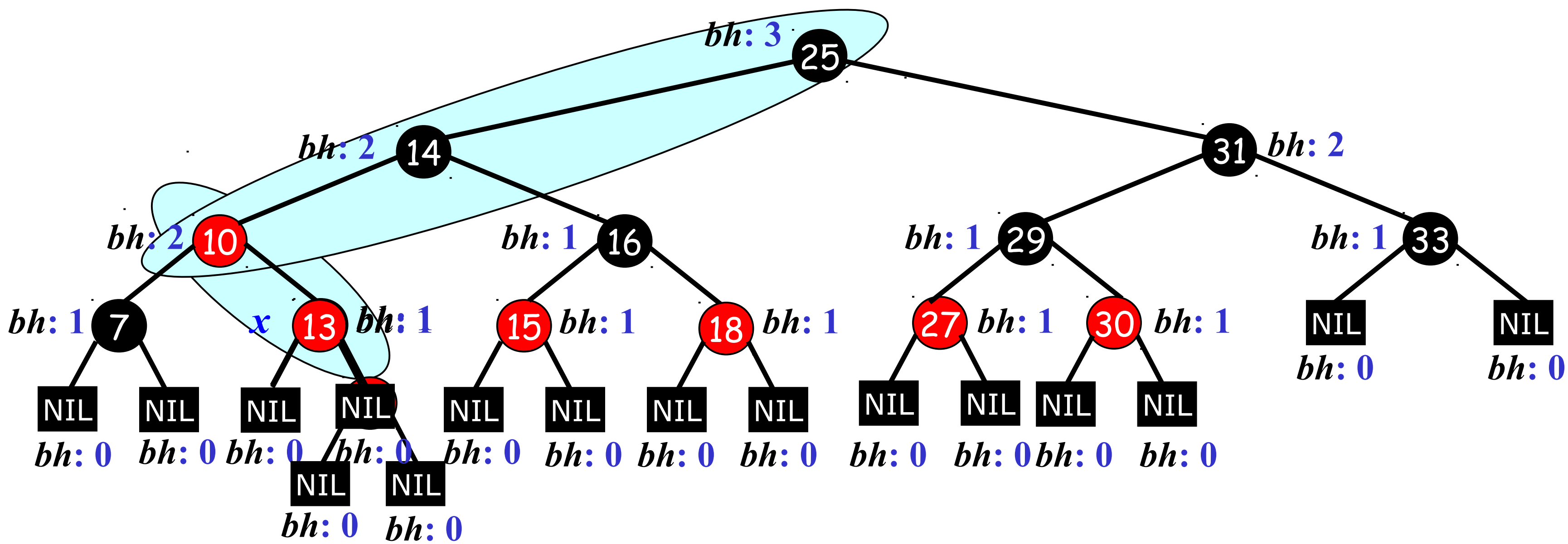
Deletion in a RB-Tree: Example

- The red-black property will disturb only when the node deleted is BLACK
- It reduces the black-height in that path by 1
- Delete 13



Deletion in a RB-Tree: Example

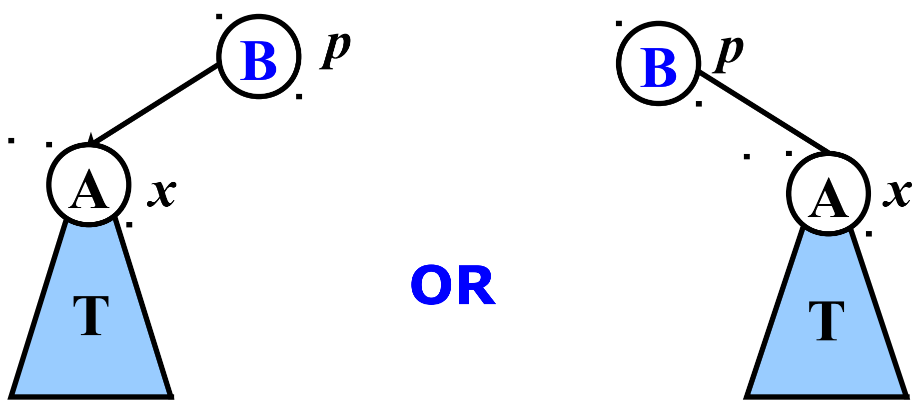
- The red-black property will disturb only when the node deleted is BLACK
- It reduces the black-height in that path by 1
- Delete 12



- Property 4 and 5 are violated

Restoring RB-Property after Deletion

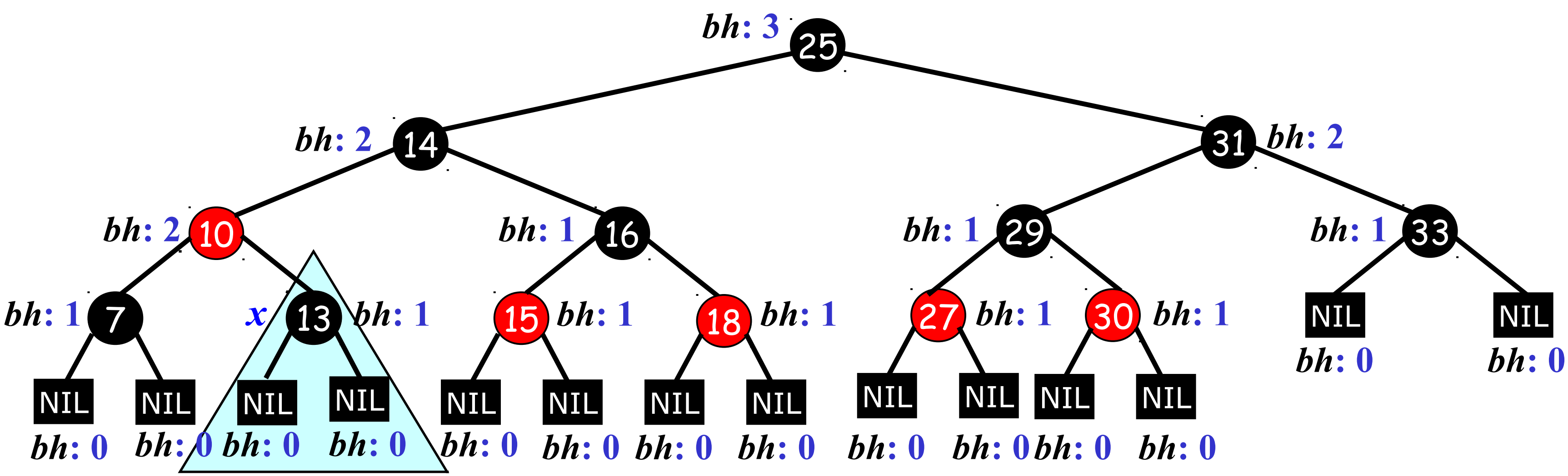
- Let T be the subtree whose black-height is reduced after deleting a BLACK node from it
- Let x be the root of the subtree T
- T is connected to the parent (p) of the deleted node
  - The parent node p is either RED or BLACK
  - T is either a left subtree or right subtree



- **If x is RED, then simply change the color of x into BLACK**

Deletion in a RB-Tree: Example

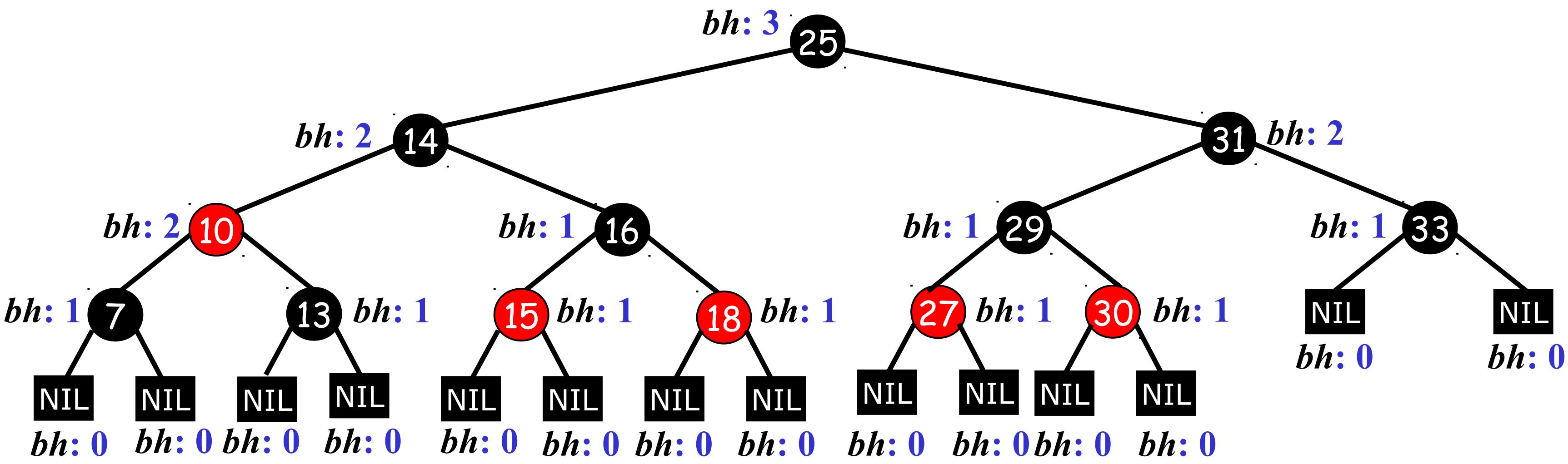
- RB-Tree after deleting 12 (12 was deleted in slide number 47)



- Since x is RED, recolor x to BLACK

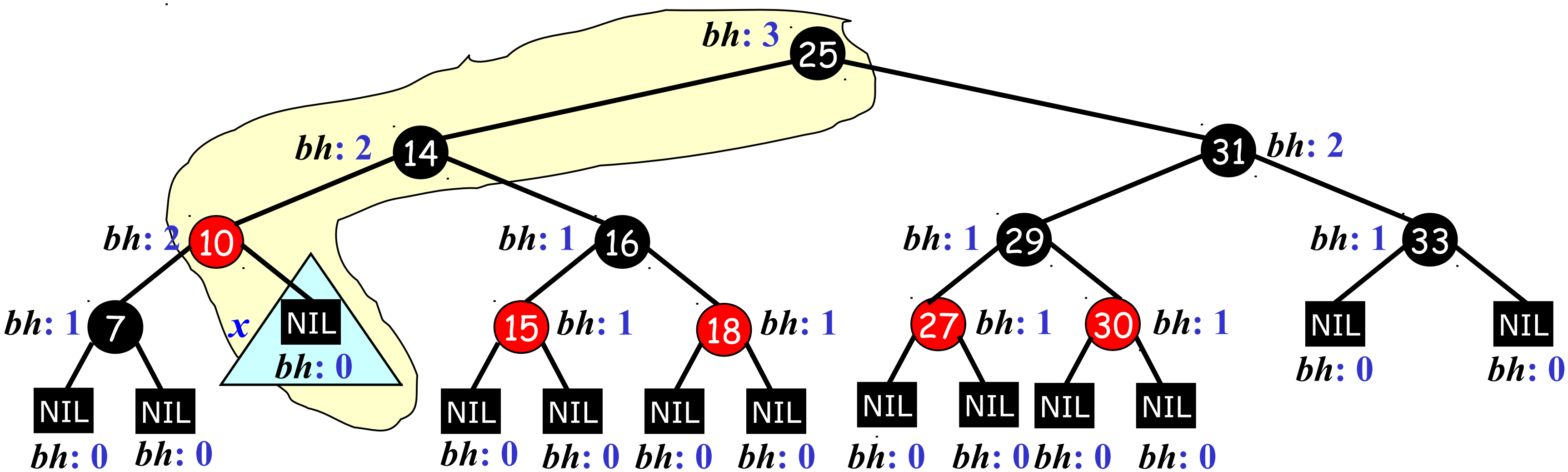
Deletion in a RB-Tree: Example

- Delete 13



Deletion in a RB-Tree: Example

- Delete 13



- Black-height property in the marked path is violated
  - In this path black-height is reduced by 1

Pseudo-code for Delete and Restoring RB-property

RB-DELETE(z)

// z is the node to be inserted

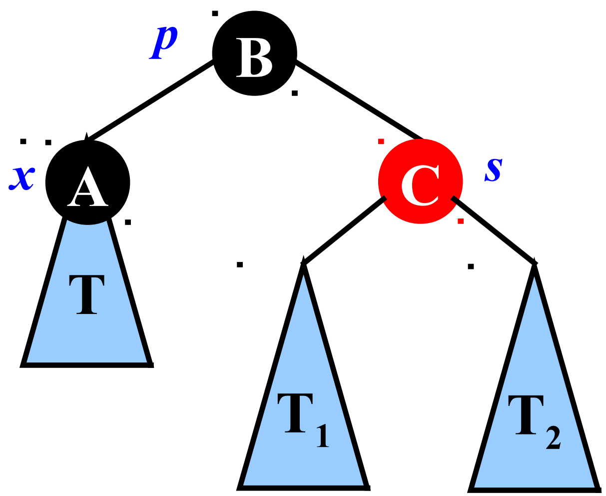
1. BST-DELETE(z)
2. Let x be the new child of parent of deleted node
3. RB-DELETE-FIXUP(x)   // To restore RB-properties

RB-DELETE-FIXUP(x)

1. while ($x \longrightarrow parent \neq$ EN-NIL) and ($x \longrightarrow color ==$ BLACK)
2.   // CASE 1 or CASE 2 or CASE 3 or CASE 4 or
       // (CASE 5 and) CASE 6 or (CASE 7 and) CASE 8
       // Refer CORMEN Book
3. end
4. $x \longrightarrow color =$ BLACK   // x is either root or color of x is RED

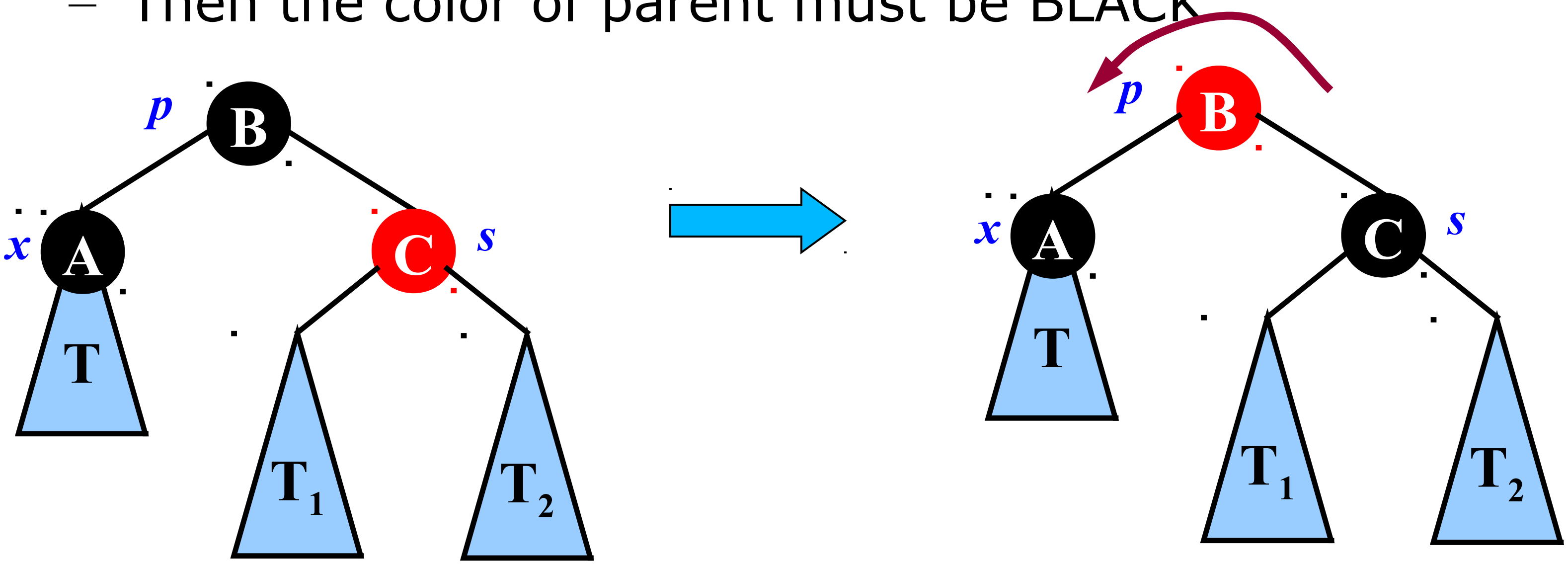
Different Cases to Restore RB-Property after Deletion

- When x is BLACK. there are 8 cases
- **CASE 1**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is RED
  - Then the color of parent must be BLACK



Different Cases to Restore RB-Property after Deletion

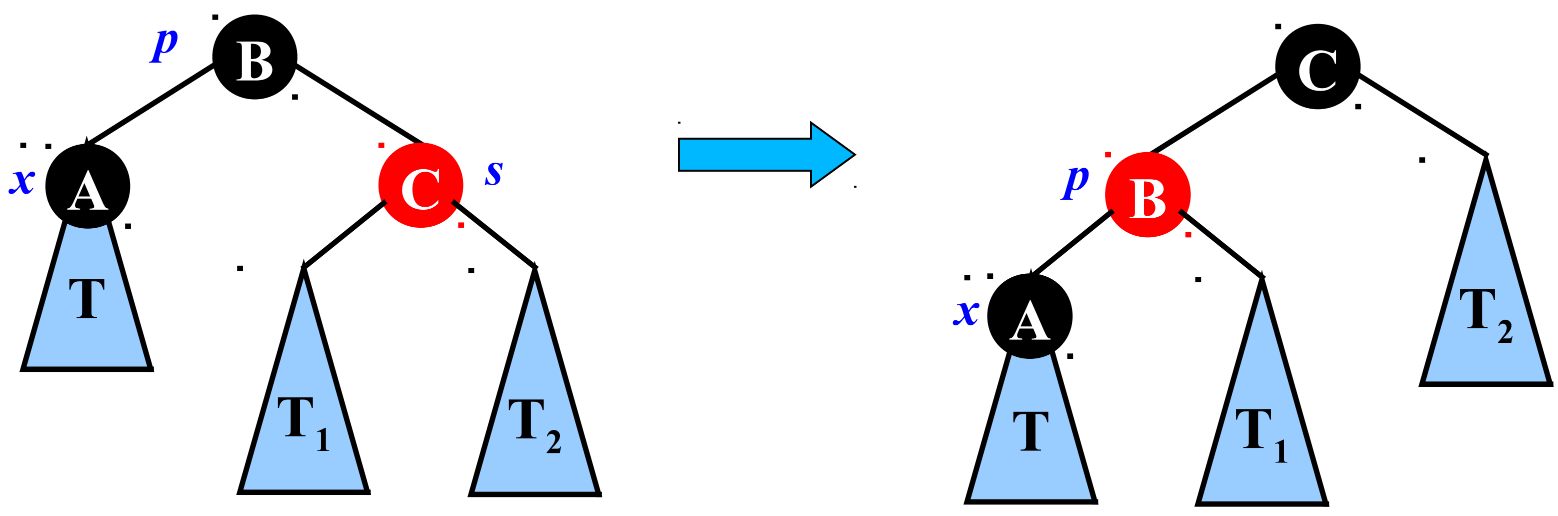
- **CASE 1**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is RED
  - Then the color of parent must be BLACK



- Recolor the sibling (s) of x to BLACK
- Recolor the parent (p) of x to RED
- Do left-rotation (LL) at parent (p) of x

Different Cases to Restore RB-Property after Deletion

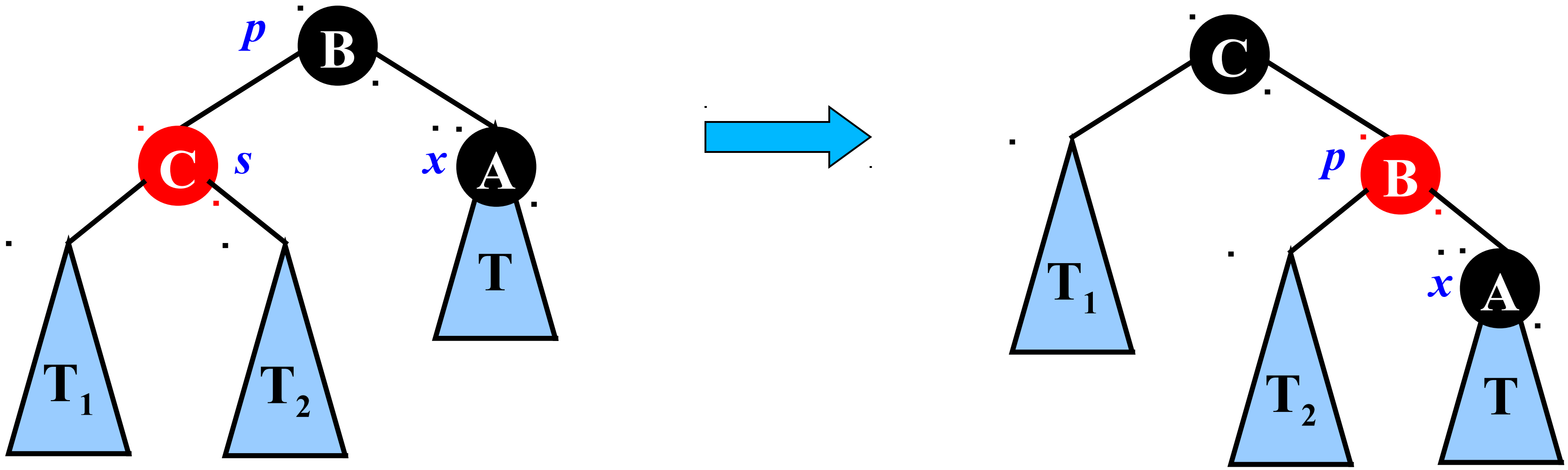
- **CASE 1**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is RED
  - Then the color of parent must be BLACK



- Recolor the sibling (s) of x to BLACK
- Recolor the parent (p) of x to RED
- Do left-rotation (LL) at parent (p) of x

Different Cases to Restore RB-Property after Deletion

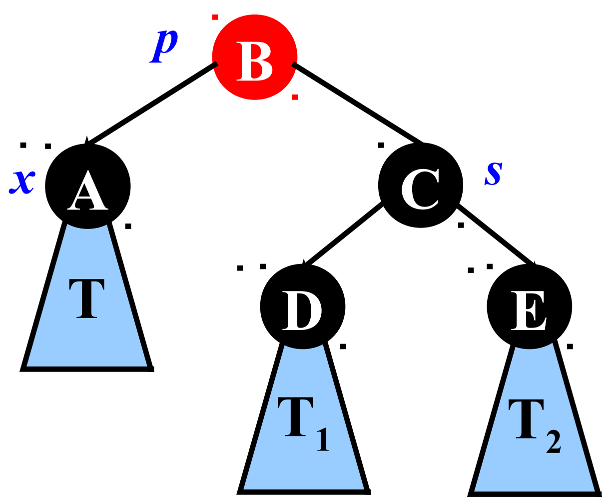
- **CASE 2**: Mirror of CASE 1
  - The x is black and is **right child** of its parent
  - The color of sibling (s) of x is RED
  - Then the color of parent must be BLACK



- Recolor the sibling (s) of x to BLACK
- Recolor the parent (p) of x to RED
- Do right-rotation (RR) at parent (p) of x

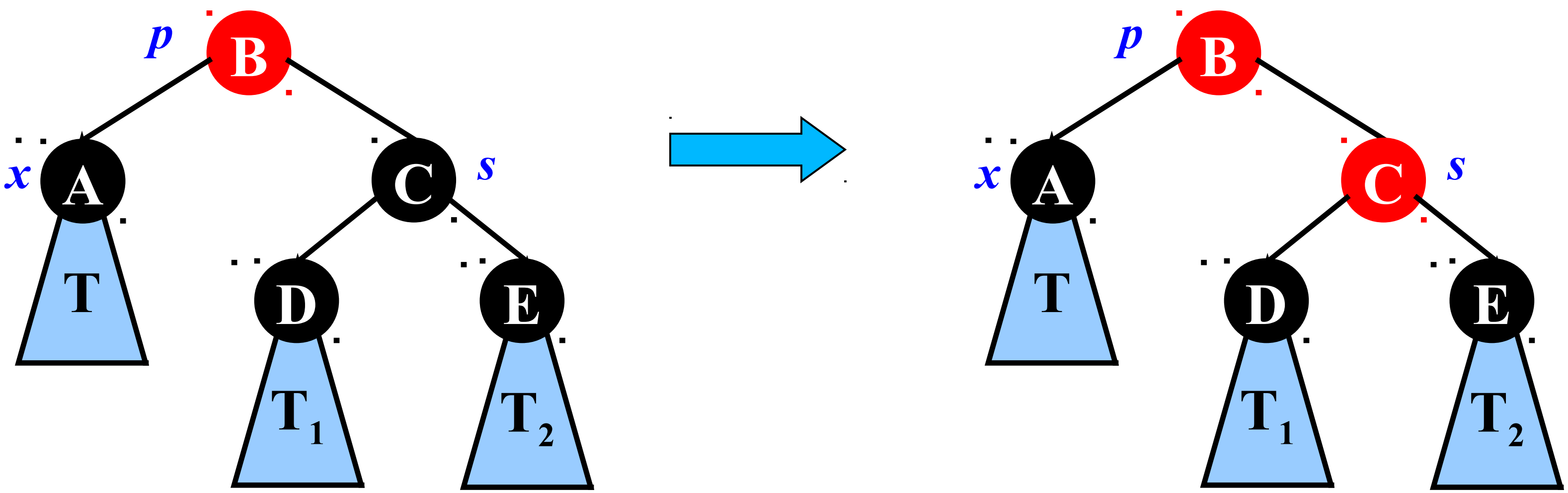
Different Cases to Restore RB-Property after Deletion

- **CASE 3**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black and both its children are black
  - The color of parent may be RED or BLACK



Different Cases to Restore RB-Property after Deletion

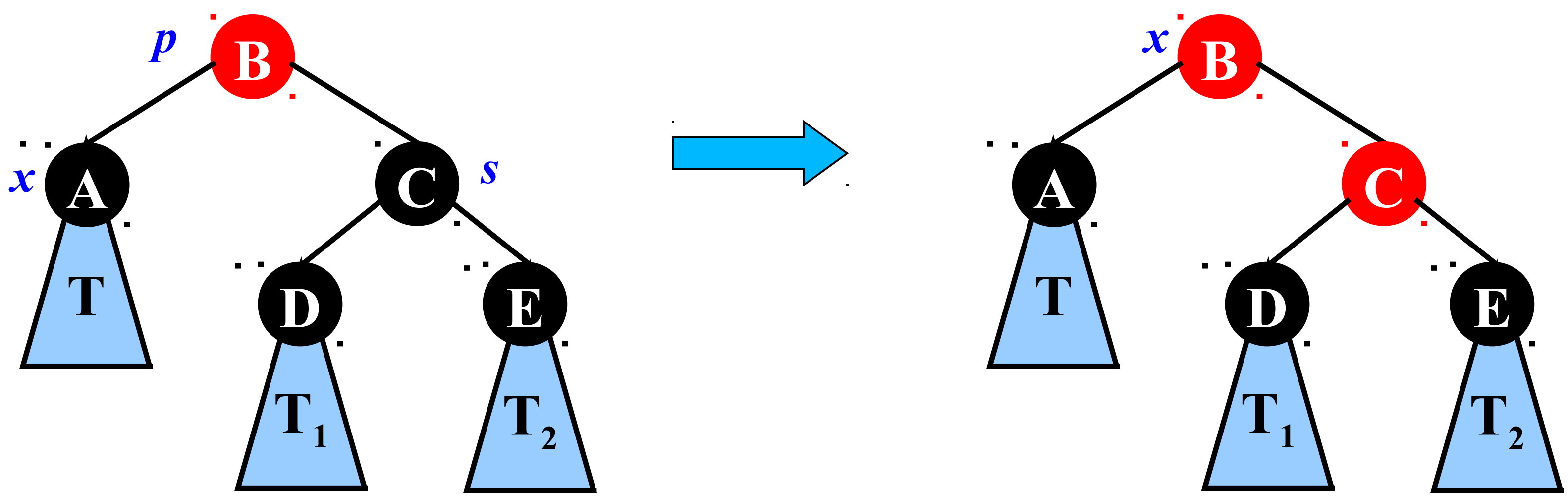
- **CASE 3**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black and both its children are black
  - The color of parent may be RED or BLACK



- Recolor the sibling (s) of x to RED

Different Cases to Restore RB-Property after Deletion

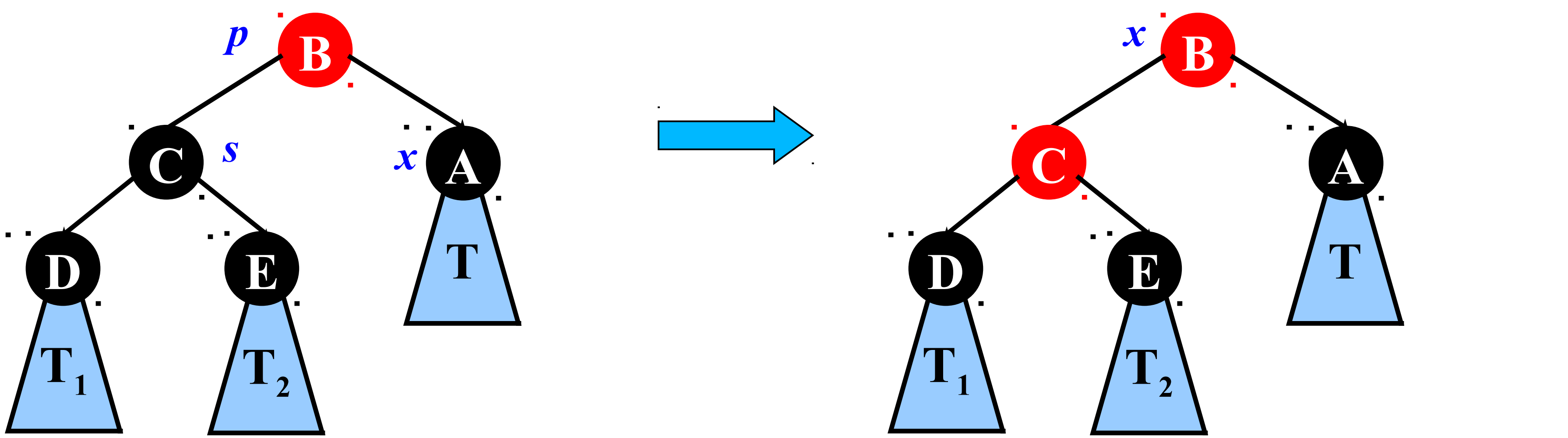
- **CASE 3**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black and both its children are black
  - The color of parent may be RED or BLACK



- Recolor the sibling (s) of x to RED
- Make parent (p) of x as new x

Different Cases to Restore RB-Property after Deletion

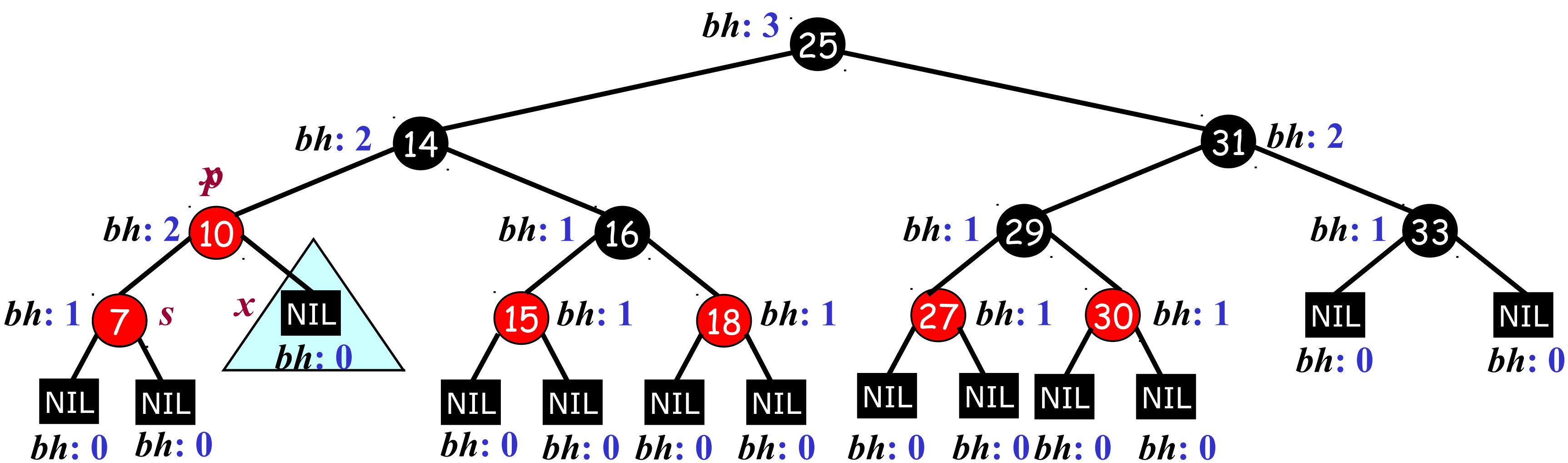
- **CASE 4**: Mirror of CASE 3
  - The x is black and is **right child** of its parent
  - The color of sibling (s) of x is black and both its children are black
  - The color of parent may be RED or BLACK



- Recolor the sibling (s) of x to RED
- Make parent (p) of x as new x

Deletion in a RB-Tree: Example

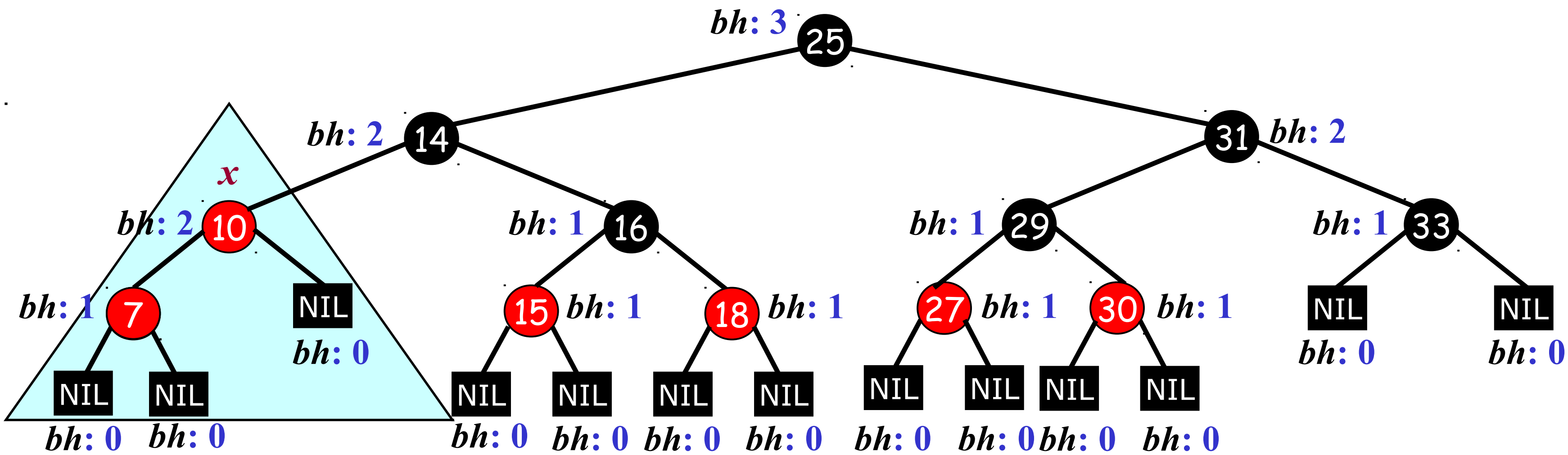
- Resulting RB-Tree after the deleting 13



- CASE 4:
  - Recolor the sibling (s) of x to RED
  - Make parent (p) of x as new x

Deletion in a RB-Tree: Example

- Resulting RB-Tree after the performing CASE 4

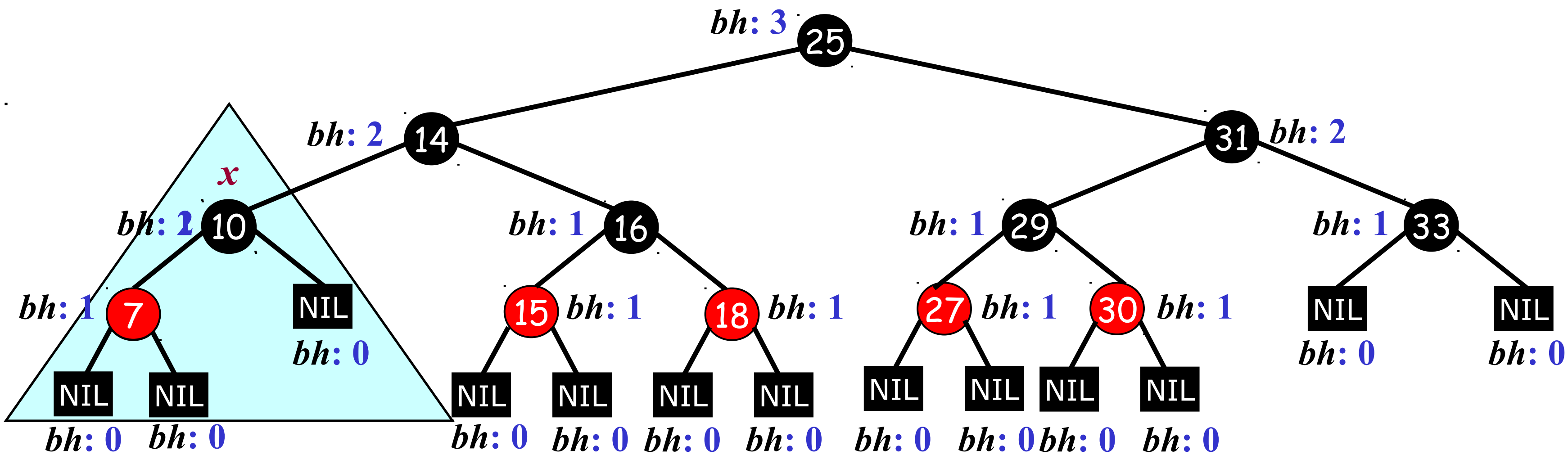


RB-DELETE-FIXUP(x)

1. while ($x \longrightarrow parent \neq$ EN-NIL) and ($x \longrightarrow color ==$ BLACK)
2. // CASE 1 or CASE 2 or CASE 3 or **CASE 4** or
   // (CASE 5 and CASE 6) or (CASE 7 and CASE 8)
   // Refer CORMEN Book
3. end
4. $x \longrightarrow color =$ BLACK // x is either root or color of x is RED

Deletion in a RB-Tree: Example

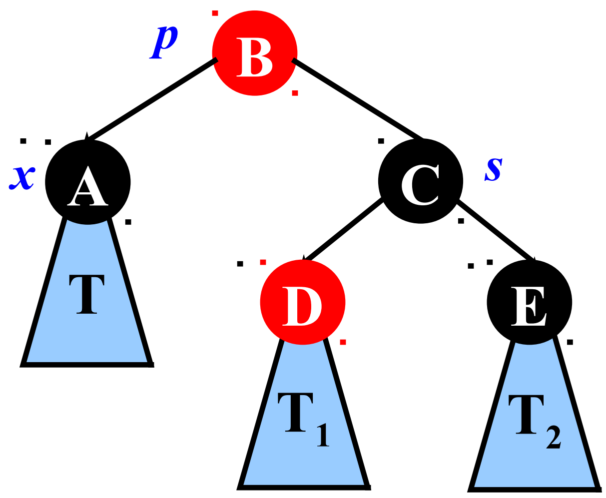
- Resulting RB-Tree after the performing CASE 4



- x is RED: Recolor x to BLACK
- Compute black-height at x

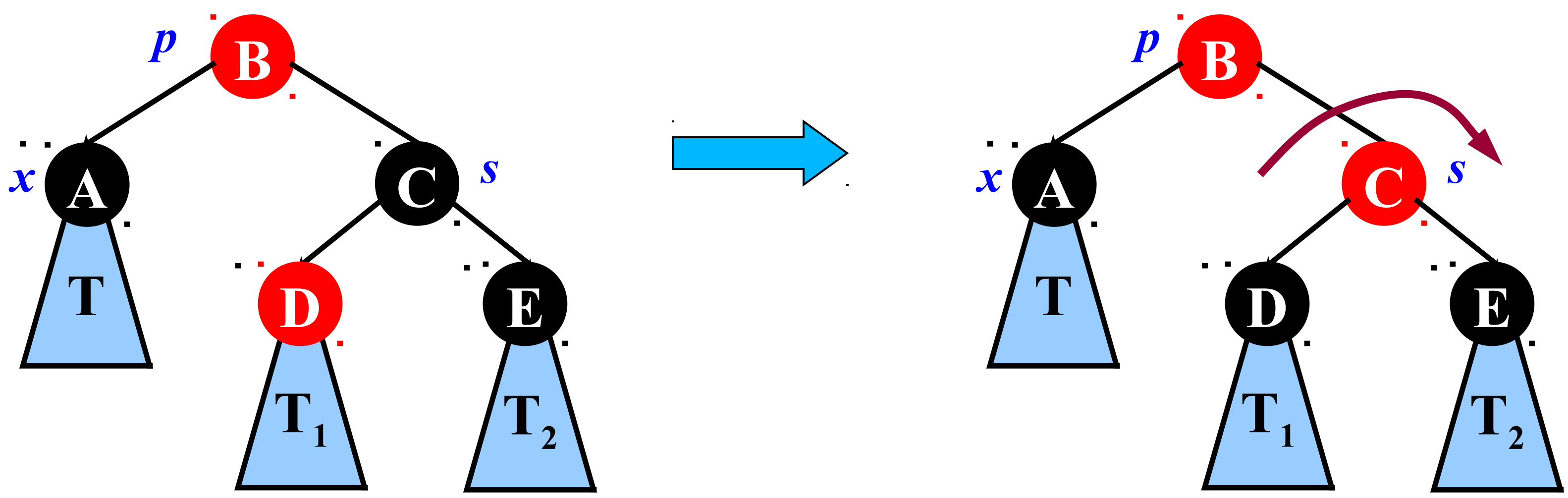
Different Cases to Restore RB-Property after Deletion

- **CASE 5**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black
  - Sibling's left child is RED and right child is BLACK
  - The color of parent may be RED or BLACK



Different Cases to Restore RB-Property after Deletion

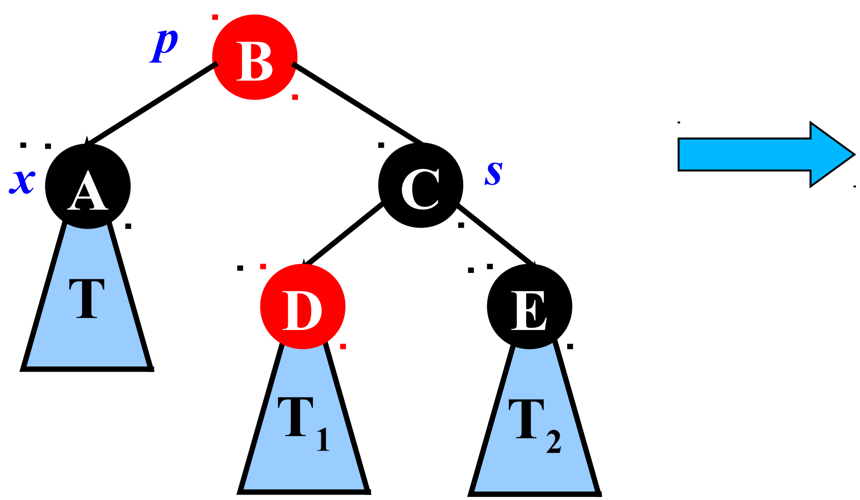
- **CASE 5**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black
  - Sibling's left child is RED and right child is BLACK
  - The color of parent may be RED or BLACK

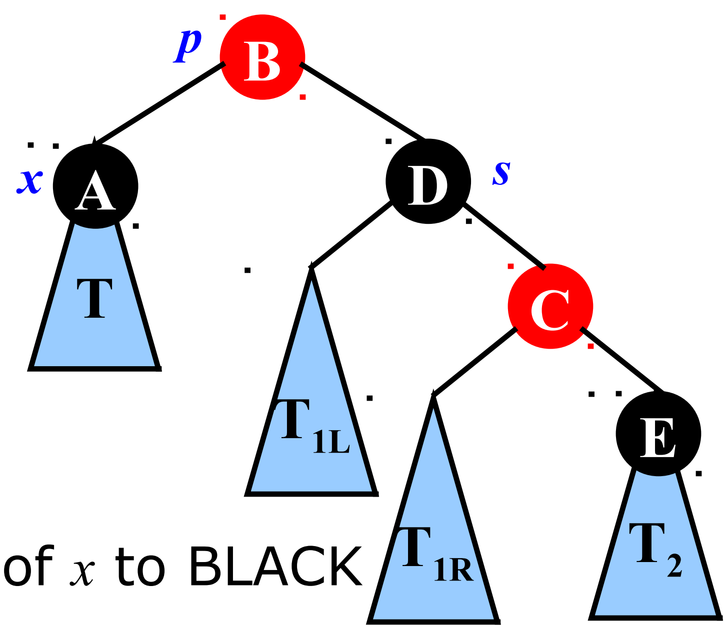


- Recolor the left child of sibling (s) of x to BLACK
- Recolor the sibling (s) of x to RED
- Do right-rotation (RR) at sibling (s) of x

Different Cases to Restore RB-Property after Deletion

- **CASE 5**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black
  - Sibling's left child is RED and right child is BLACK
  - The color of parent may be RED or BLACK

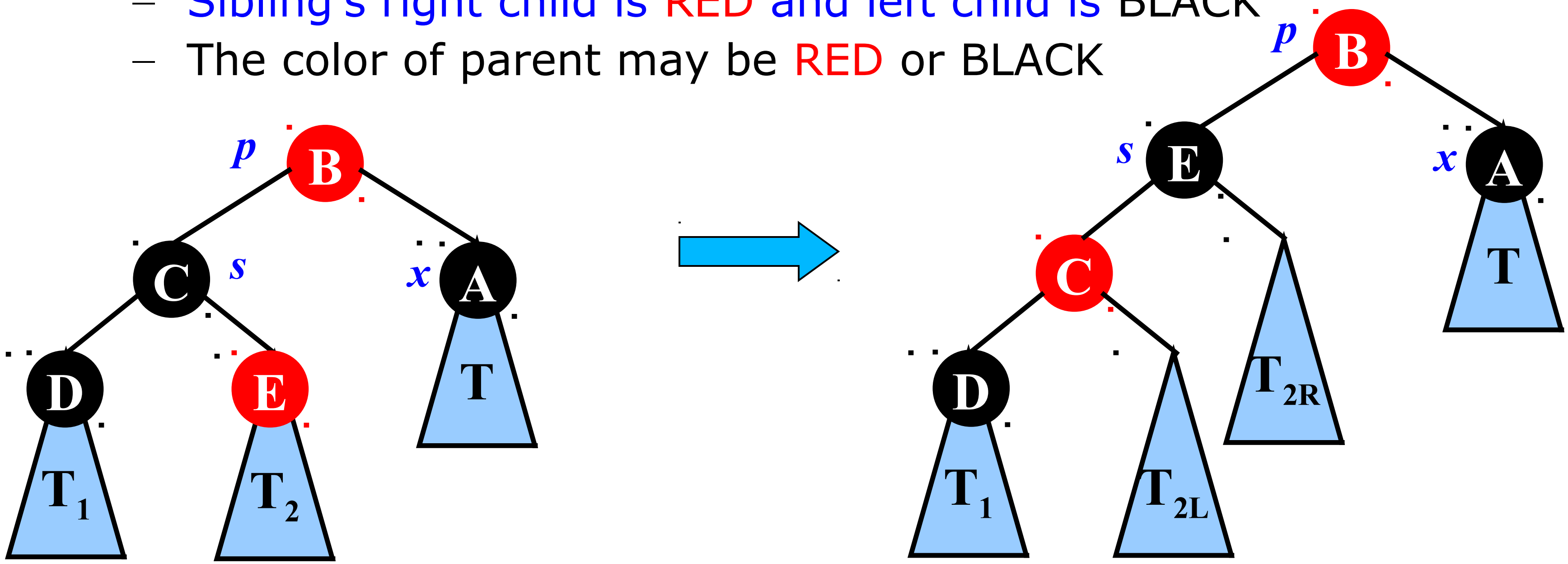




- Recolor the left child of sibling (s) of x to BLACK
- Recolor the sibling (s) of x to RED
- Do right-rotation (RR) at sibling (s) of x
- **CASE 5 leads to CASE 6**

Different Cases to Restore RB-Property after Deletion

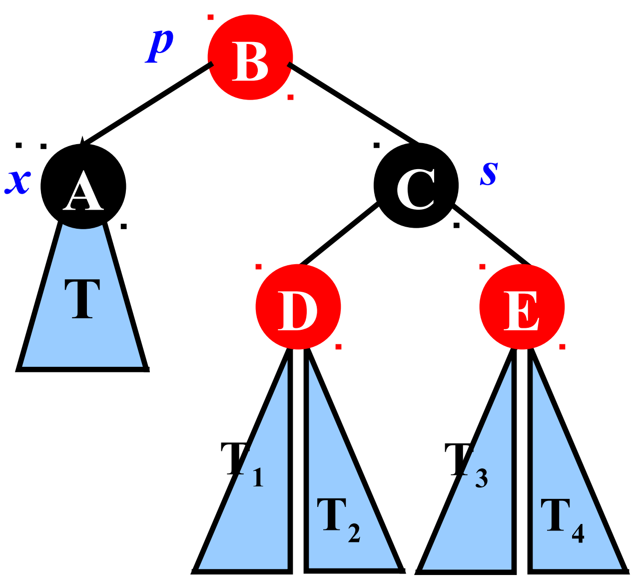
- **CASE 7**: Mirror of CASE 5
  - The x is black and is **right child** of its parent
  - The color of sibling (s) of x is black
  - Sibling's right child is RED and left child is BLACK
  - The color of parent may be RED or BLACK



- Recolor the right child of sibling (s) of x to BLACK
- Recolor the sibling (s) of x to RED
- Do left-rotation (LL) at sibling (s) of x
- **CASE 7 leads to CASE 8**

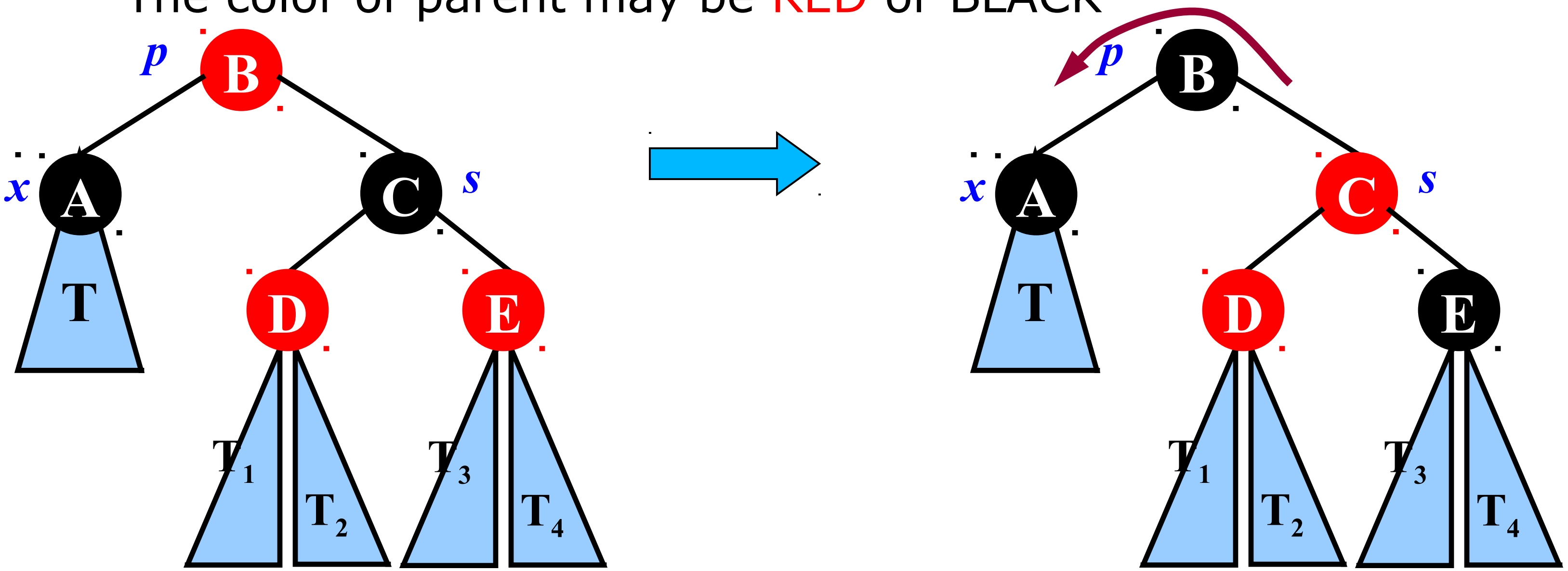
Different Cases to Restore RB-Property after Deletion

- **CASE 6**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black
  - Sibling's right child is RED (i.e. left child is RED/BLACK)
  - The color of parent may be RED or BLACK



Different Cases to Restore RB-Property after Deletion

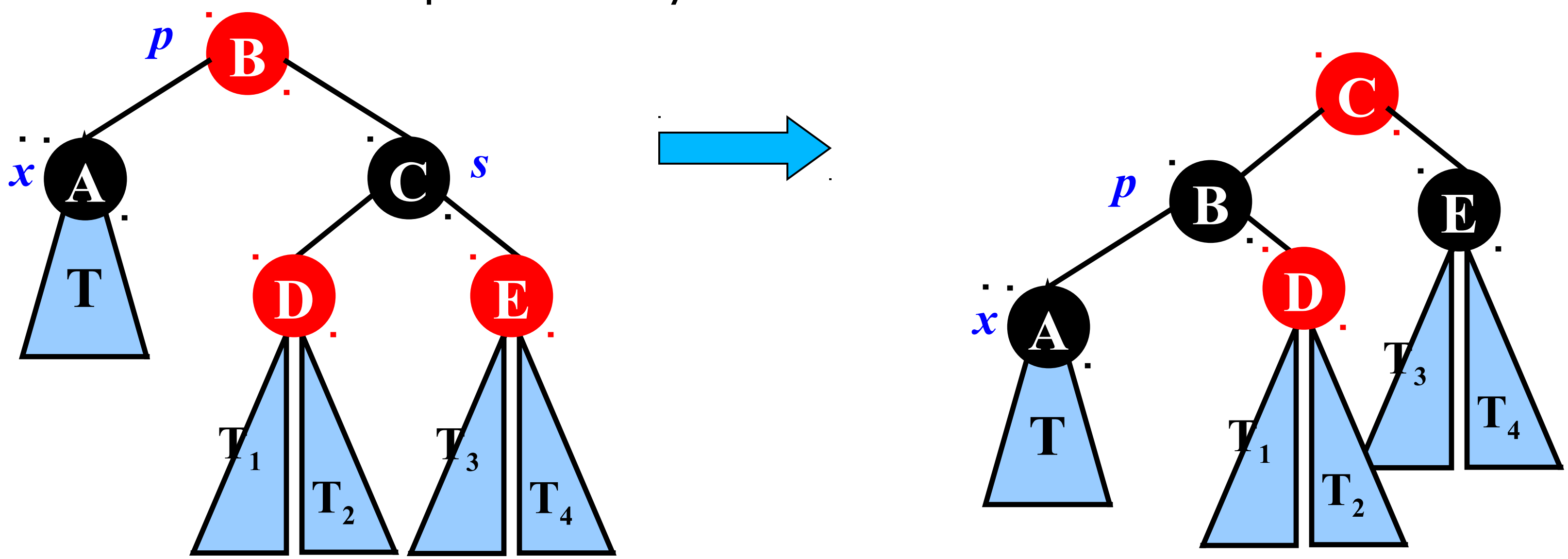
- **CASE 6**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black
  - Sibling's right child is RED (i.e. left child is RED/BLACK)
  - The color of parent may be RED or BLACK



- Recolor the sibling (s) of x with the parent (p) of x
- Recolor the parent (p) of x to BLACK
- Recolor the right child of sibling (s) of x to BLACK
- Do left-rotation (LL) at parent (p) of x
-

Different Cases to Restore RB-Property after Deletion

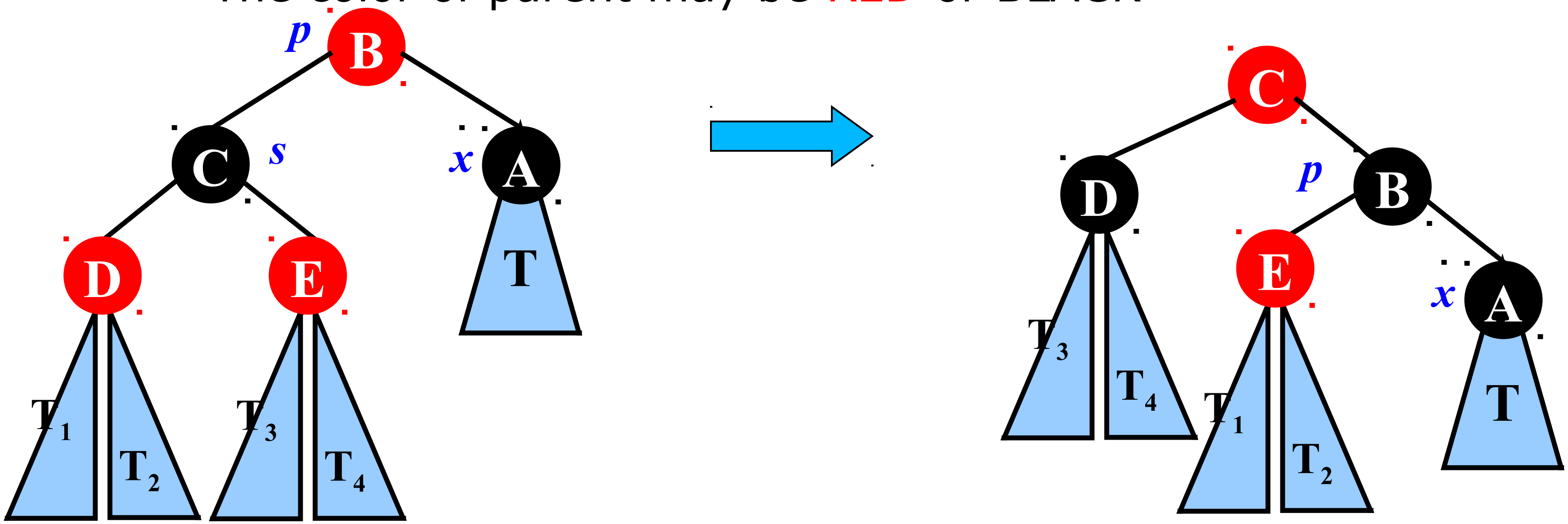
- **CASE 6**:
  - The x is black and is **left child** of its parent
  - The color of sibling (s) of x is black
  - Sibling's right child is RED (i.e. left child is RED/BLACK)
  - The color of parent may be RED or BLACK



- Recolor the sibling (s) of x with the parent (p) of x
- Recolor the parent (p) of x to BLACK
- Recolor the right child of sibling (s) of x to BLACK
- Do left-rotation (LL) at parent (p) of x
- **Set ROOT as new x**

Different Cases to Restore RB-Property after Deletion

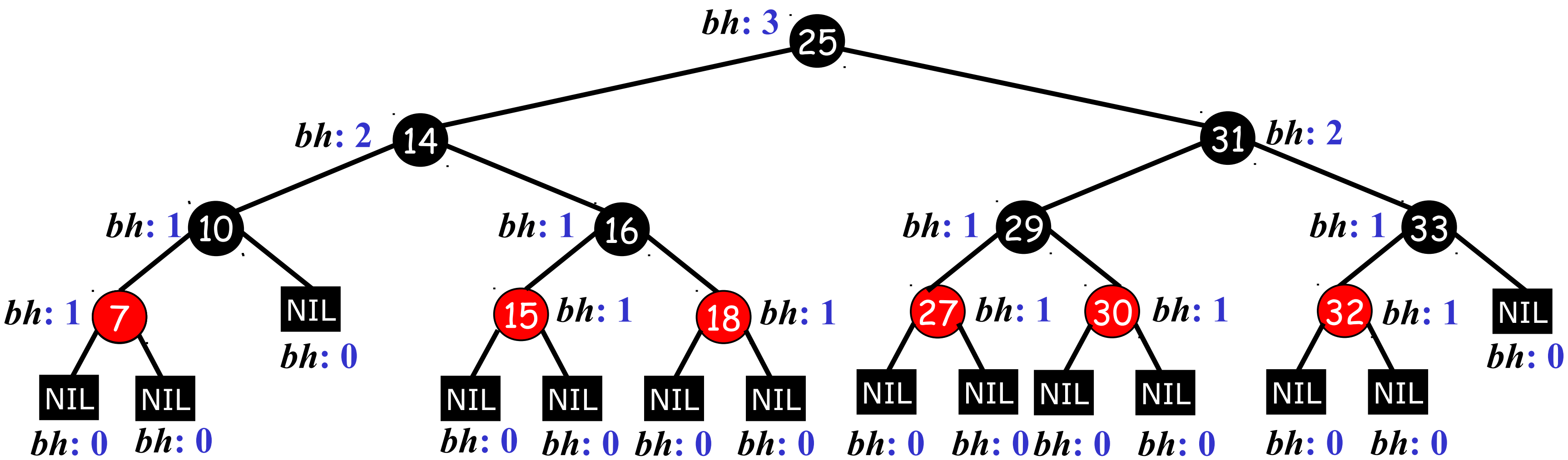
- **CASE 8**: Mirror of CASE 6
  - The x is black and is **right child** of its parent
  - The color of sibling (s) of x is black
  - Sibling's left child is RED (i.e. right child is RED/BLACK)
  - The color of parent may be RED or BLACK



- Recolor the sibling (s) of x with the parent (p) of x
- Recolor the parent (p) of x to BLACK
- Recolor the left child of sibling (s) of x to BLACK
- Do right-rotation (RR) at parent (p) of x
- **Set ROOT as new x**

Deletion in a RB-Tree: Example

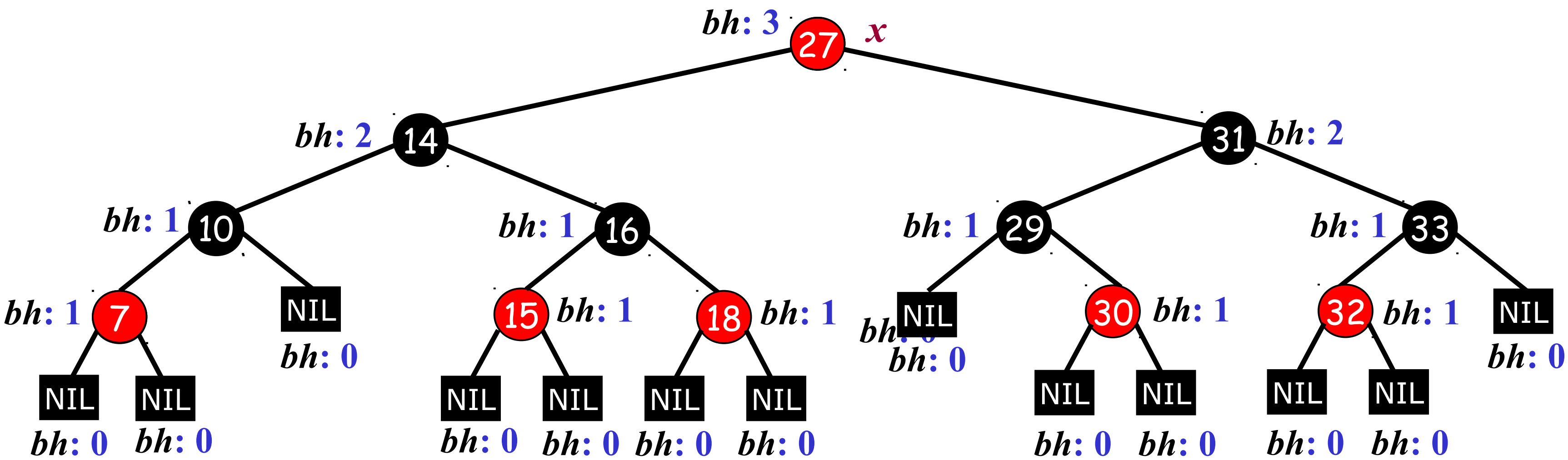
- Delete 25



- Node with key 27 is the successor
- Delete the successor
- Copy the content of successor to node with key 25

Deletion in a RB-Tree: Example

- Delete 25

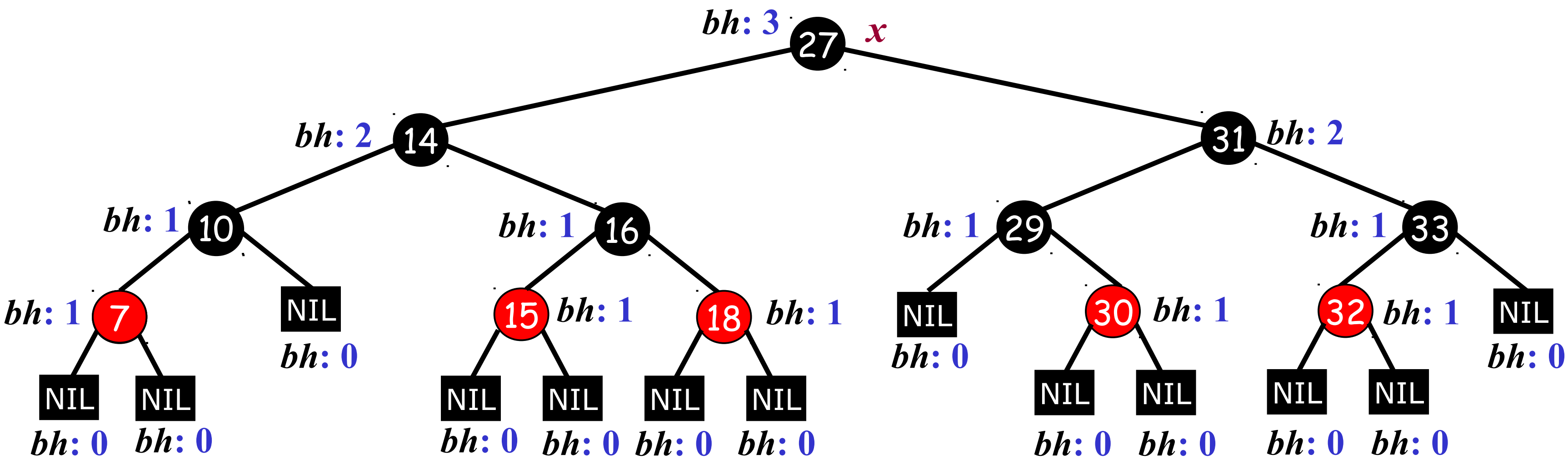


RB-DELETE-FIXUP(x)

1. while ($x \longrightarrow parent \neq$ EN-NIL) and ($x \longrightarrow color$ = BLACK)
2. // CASE 1 or CASE 2 or CASE 3 or **CASE 4** or
   // (CASE 5 and CASE 6) or (CASE 7 and CASE 8)
   // Refer CORMEN Book
3. end
4. $x \longrightarrow color$ = BLACK // x is either root or color of x is RED

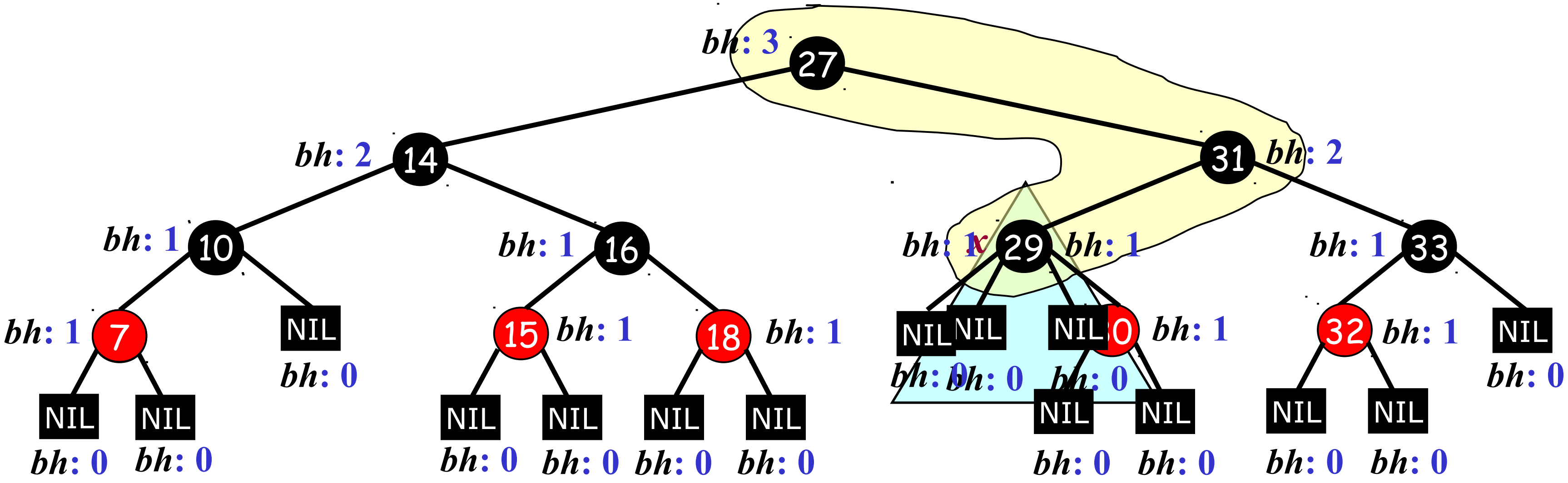
Deletion in a RB-Tree: Example

- Resulting RB-Tree after the performing recoloring



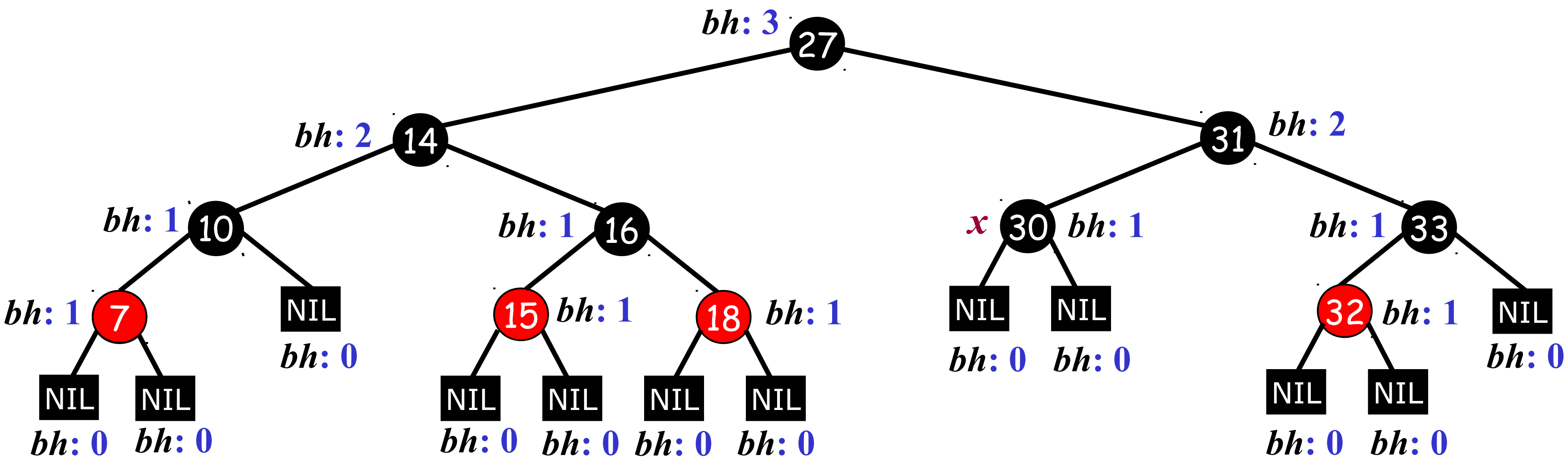
Deletion in a RB-Tree: Example

- Delete 29



Deletion in a RB-Tree: Example

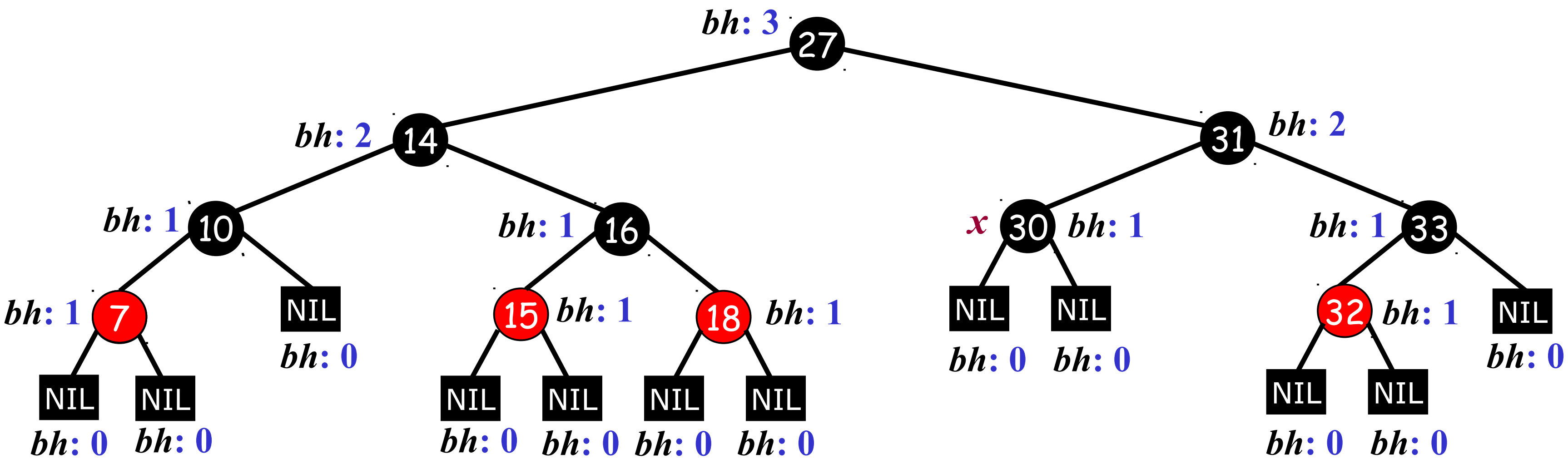
- Balanced RB-Tree after deleting 29



Since x is RED, recolor x to BLACK

Deletion in a RB-Tree: Example

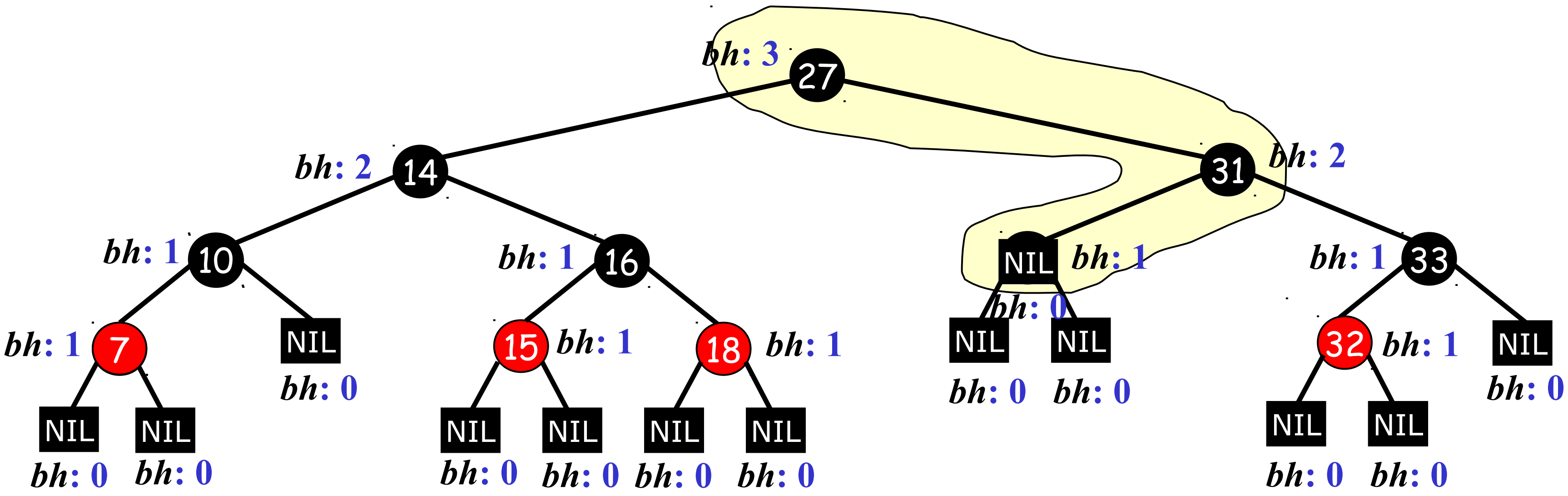
- Delete 30



Since x is RED, recolor x to BLACK

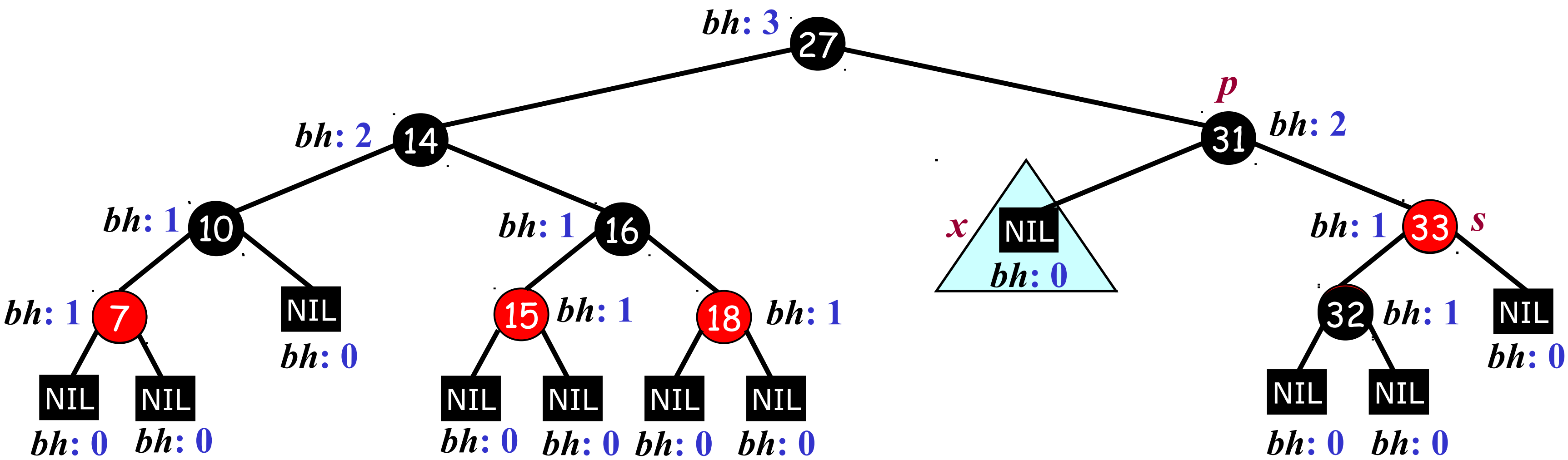
Deletion in a RB-Tree: Example

- Delete 30



Deletion in a RB-Tree: Example

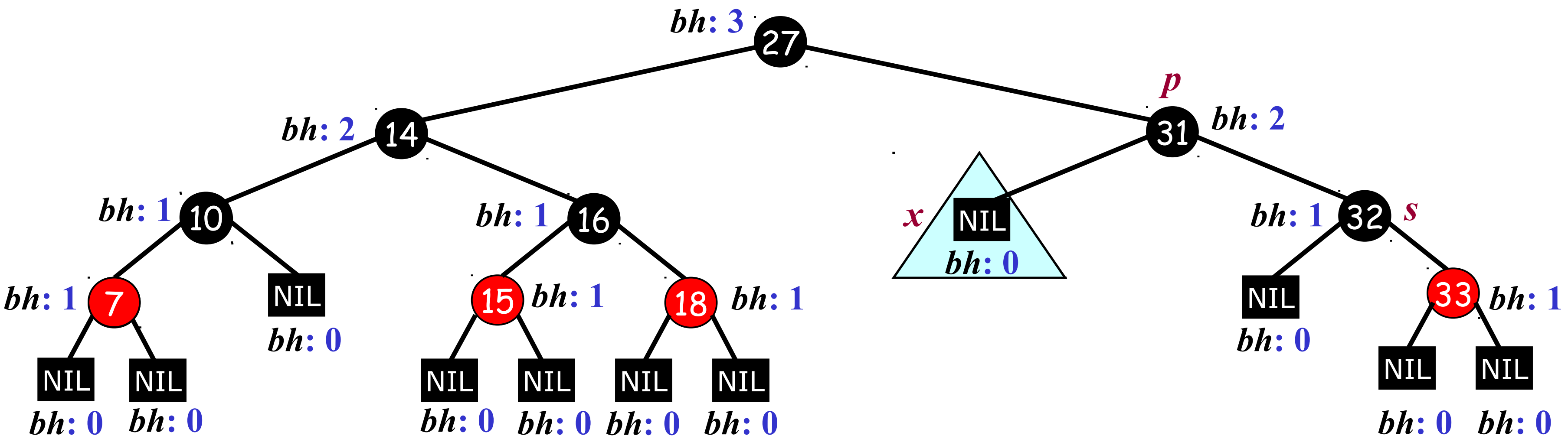
- Delete 30



- CASE 5:
  - Recolor the left child of sibling (s) of x to BLACK
  - Recolor the sibling (s) of x to RED
  - Do right-rotation (RR) at sibling (s) of x

Deletion in a RB-Tree: Example

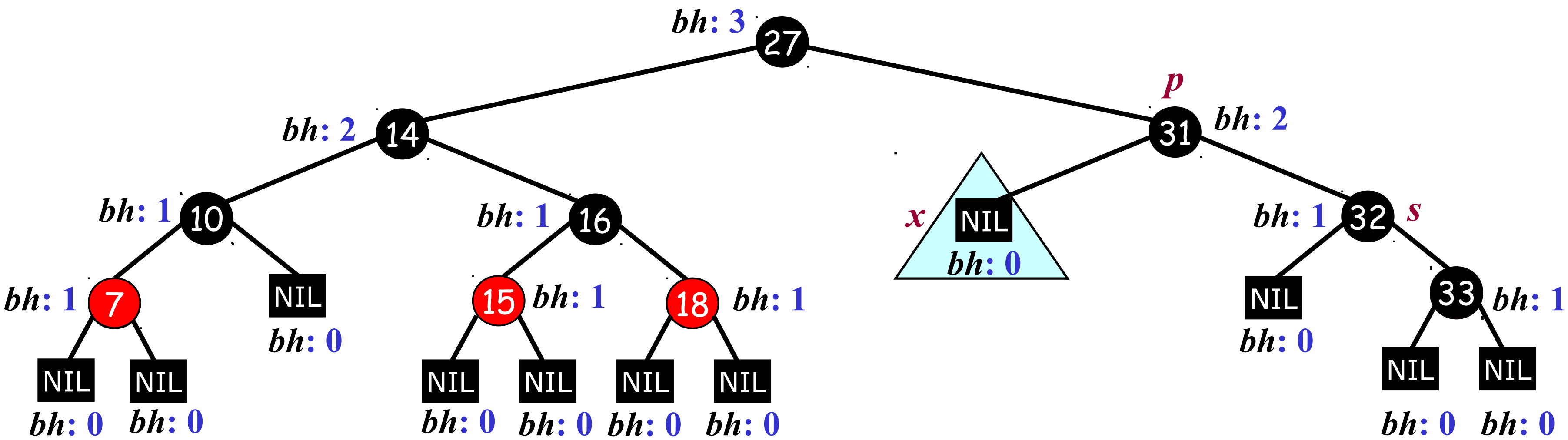
- Delete 30



- CASE 5:
  - Recolor the left child of sibling (s) of x to BLACK
  - Recolor the sibling (s) of x to RED
  - Do right-rotation (RR) at sibling (s) of x
  - **CASE 5 leads to CASE 6**

Deletion in a RB-Tree: Example

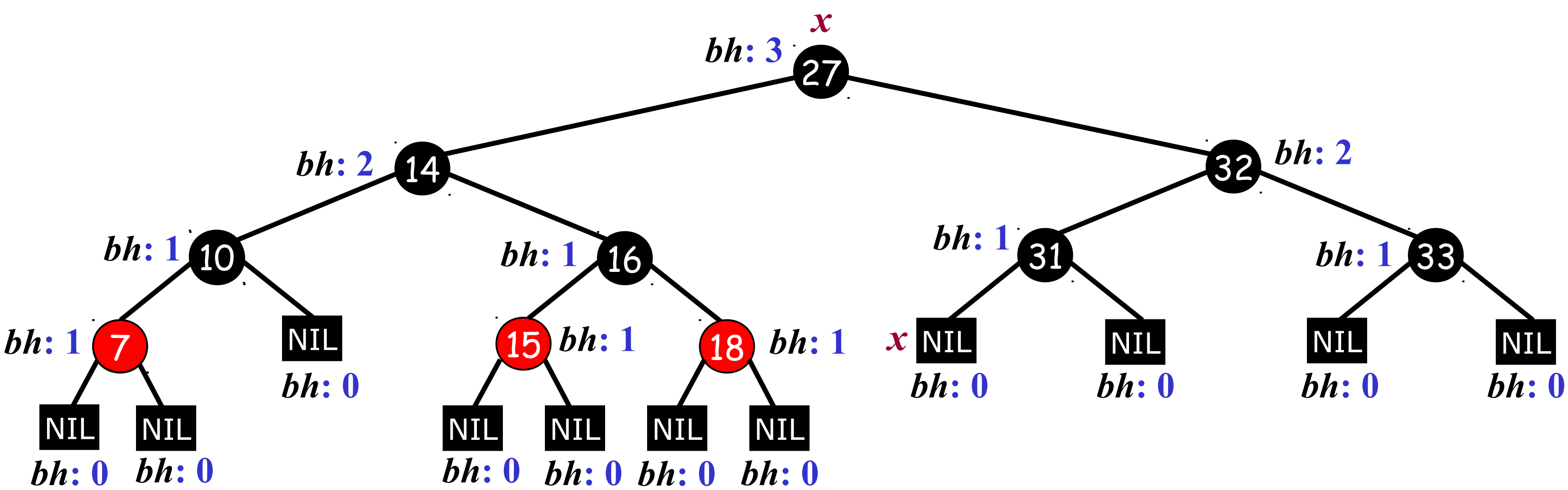
- Delete 30



- CASE 6:
  - Recolor the sibling (s) of x with the parent (p) of x
  - Recolor the parent (p) of x to BLACK
  - Recolor the right child of sibling (s) of x to BLACK
  - Do left-rotation (LL) at parent (p) of x

Deletion in a RB-Tree: Example

- Delete 30



- CASE 6:
  - Recolor the sibling (s) of x with the parent (p) of x
  - Recolor the parent (p) of x to BLACK
  - Recolor the right child of sibling (s) of x to BLACK
  - Do left-rotation (LL) at parent (p) of x
  - **Set ROOT as new x**

Use of RB-Tree

- RB-tree is common in Linux kernel
  - Implementation of completely fair scheduler
  - To keep track of virtual memory segments for a process
- Used in Java HashMap