CS 313: Intermediate Data Structure
Project 4: Red Black Tree

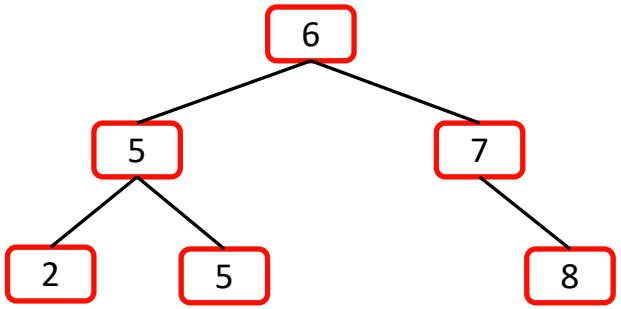
Viet Lai

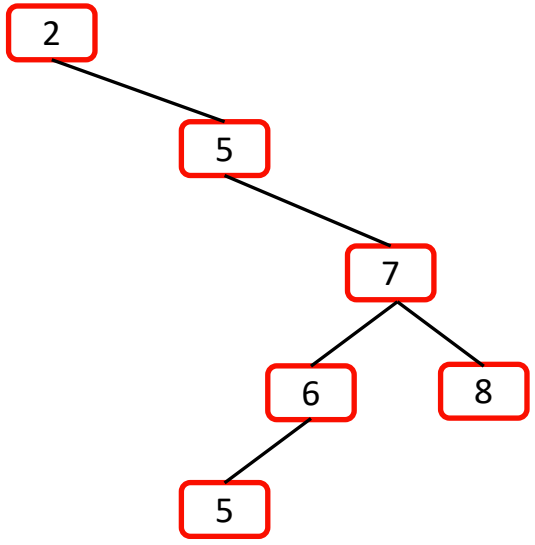
vietl@uoregon.edu

What is a binary search tree?

- Binary search tree's properties
  - Binary tree
  - Left child <= Parent
  - Right child >= Parent

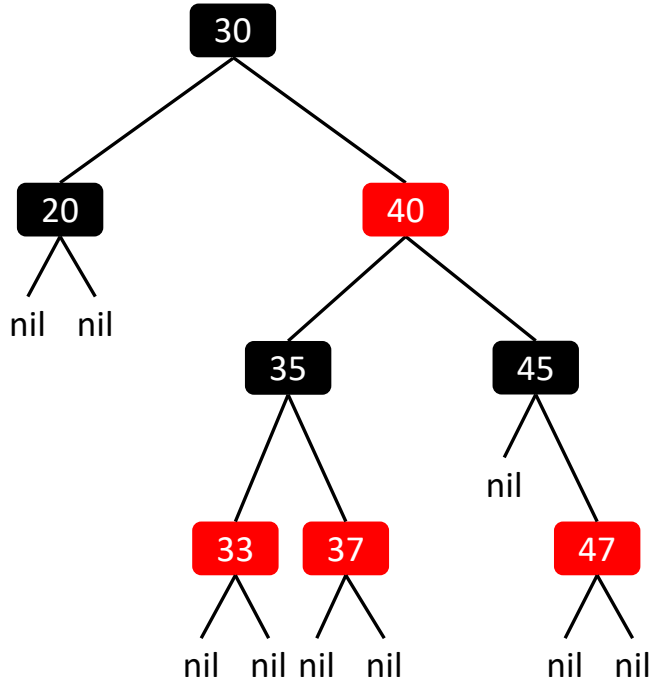
- Given a tree of N nodes, what is the depth of the tree?
- Which tree is more efficient?





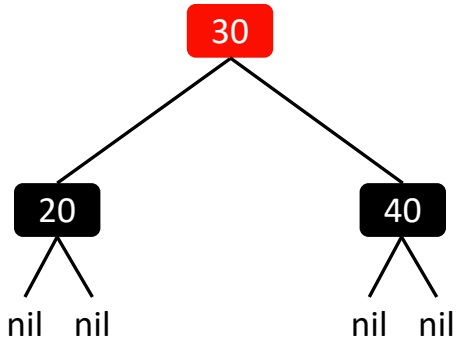
Red Black Tree

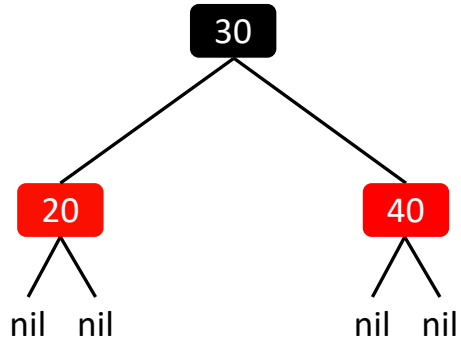
- Red Black Tree is a type of balanced binary tree
  - The leaves are NIL
- Properties
  - P1: A node is either red or black.
  - P2: The root and (NIL) leaves are black.
  - P3: If a node is red, its children are black.
  - P4: All paths from a node to all its NIL descendants contain the same number of black nodes.

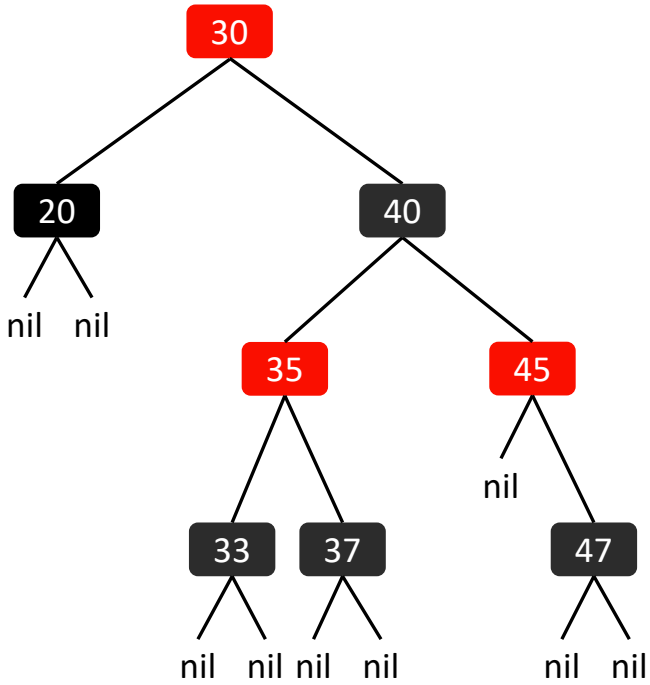


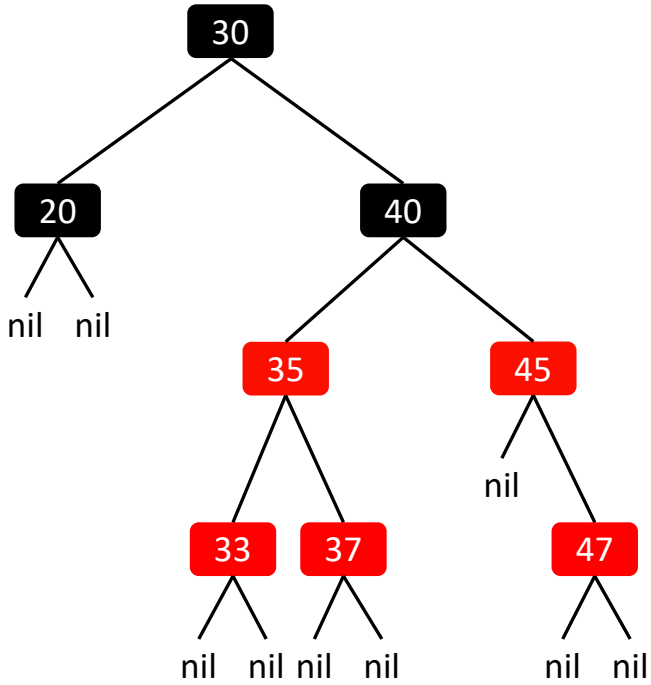
Which one is RBTree?

- Properties
  - P1: A node is either red or black.
  - P2: The root and (NIL) leaves are black.
  - P3: If a node is red, its children are black.
  - P4: All paths from a node to all its NIL descendants contain the same number of black nodes.

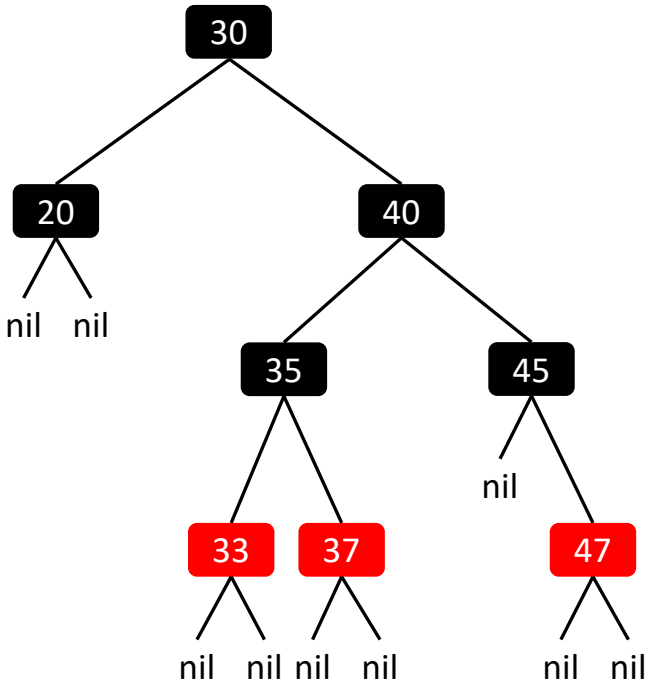








Which one is RBTree?



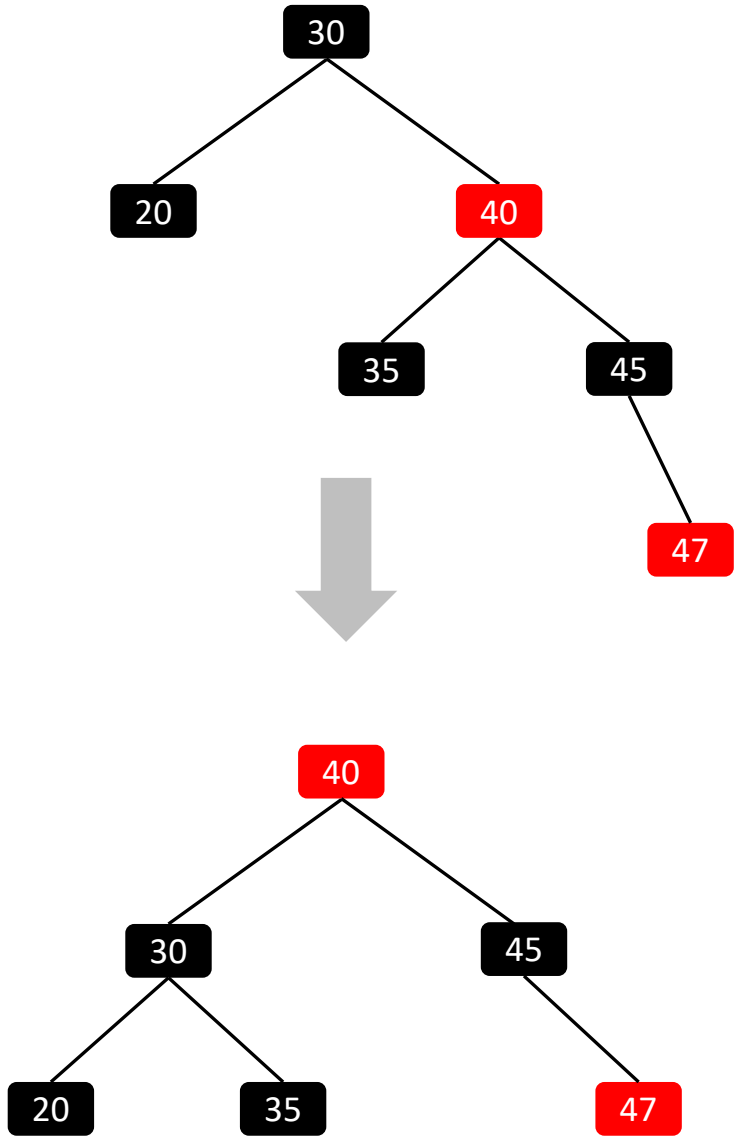
- Red Black Tree is a type of balanced binary tree
  - The leaves are NIL
- Properties
  - P1: A node is either red or black.
  - P2: The root and (NIL) leaves are black.
  - P3: If a node is red, its children are black.
  - P4: All paths from a node to all its NIL descendants contain the same number of black nodes.

Extras

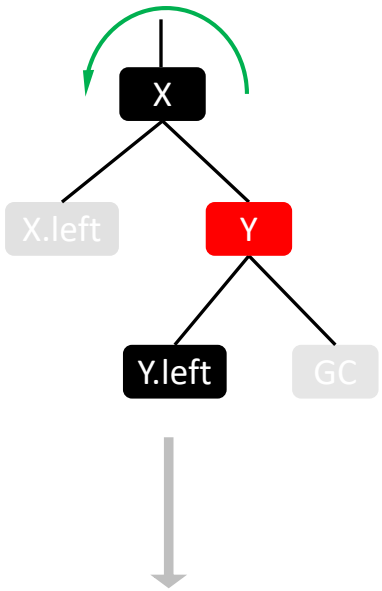
- RBNode requires 1 variable to store the color
- The longest path (root to deepest NIL) is no more than twice the length of the shortest path (root to shallowest NIL). Why?
- Which function can make the tree violate the RBTree properties?
  - Search
  - Insert
  - Delete

Rotation

- Alter structure of the tree by rearrange the subtrees
- Does not affect the order of the elements
- Goal:
  - Decrease the height of the tree
  - Larger subtree goes up
  - Smaller subtree goes down
- Side effect
  - Color changes



Left Rotation

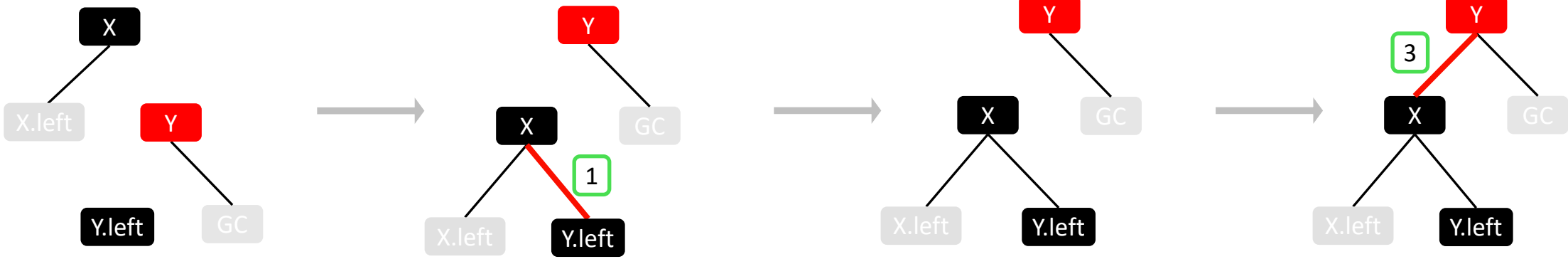


1. X and Y.left

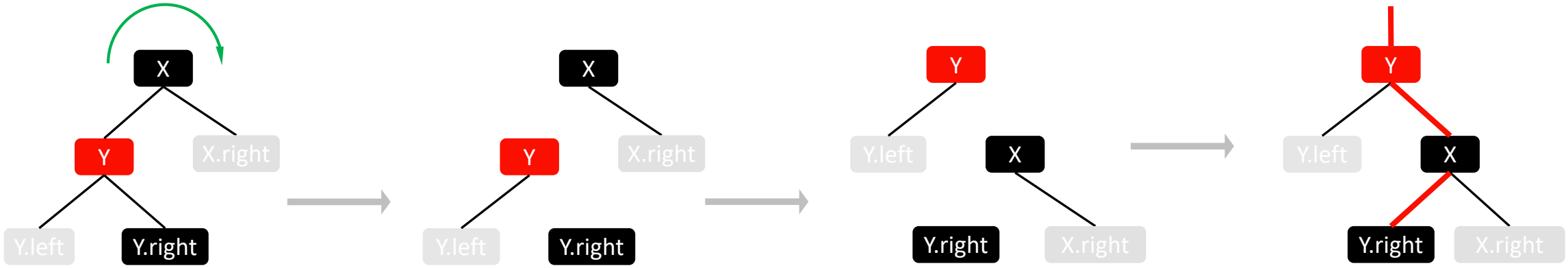
2. Y's new parent

3. X and Y

```
LEFT-ROTATE(T, x)
 1   y = x.right                  // set y
 2   x.right = y.left             // turn y's left subtree into x's right subtree
 3   if y.left ≠ T.nil
 4       y.left.p = x
 5   y.p = x.p                    // link x's parent to y
 6   if x.p == T.nil
 7       T.root = y
 8   elseif x == x.p.left
 9       x.p.left = y
10   else x.p.right = y
11   y.left = x                   // put x on y's left
12   x.p = y
```



Right Rotation



Insert Steps

- Create a new red node: Z
- Insert the red node similar to BST
- Fix the non-compliant subtree (insertFixup)

```
TREE-INSERT(T, z)
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == NIL
10      T.root = z
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
```

```
RB-INSERT(T, z)
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == T.nil
10      T.root = z
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

Set up color and children

Fix up

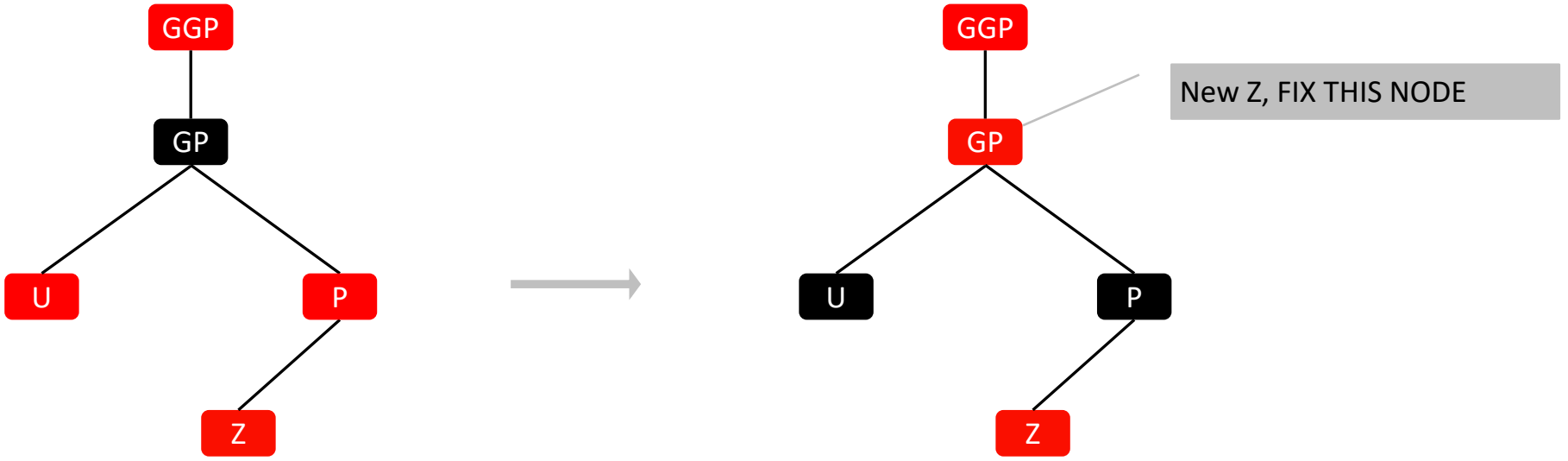
Case 0: Z is the root

- Properties
  - P1: A node is either red or black.
  - P2: The root and (NIL) leaves are black.
  - P3: If a node is red, its children are black.
  - P4: All paths from a node to all its NIL descendants contain the same number of black nodes.

- Solution:
  - Recolor Z to Z

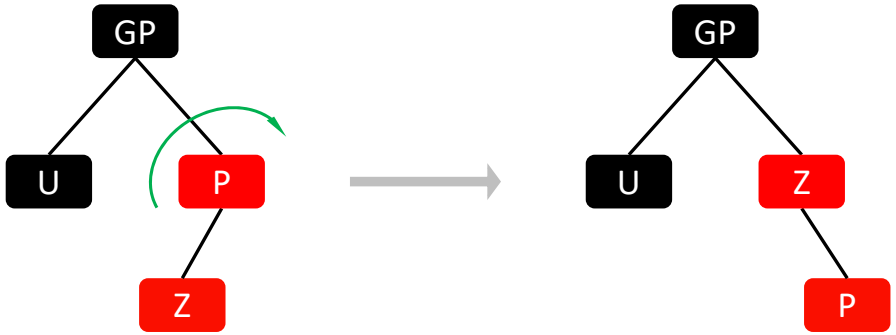
Case 1: Z has a red uncle

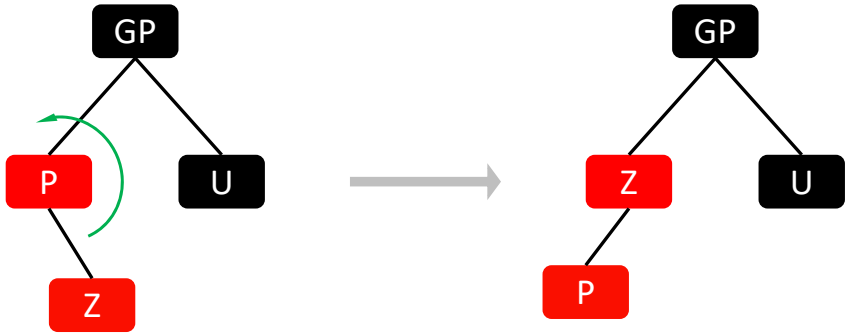
- Solution:
  - Recolor P to P
  - Recolor U to U
  - Recolor GP to GP
  - Bubble up, fix grandpa



Case 2: Z has a black uncle and GP-P-Z is a triangle

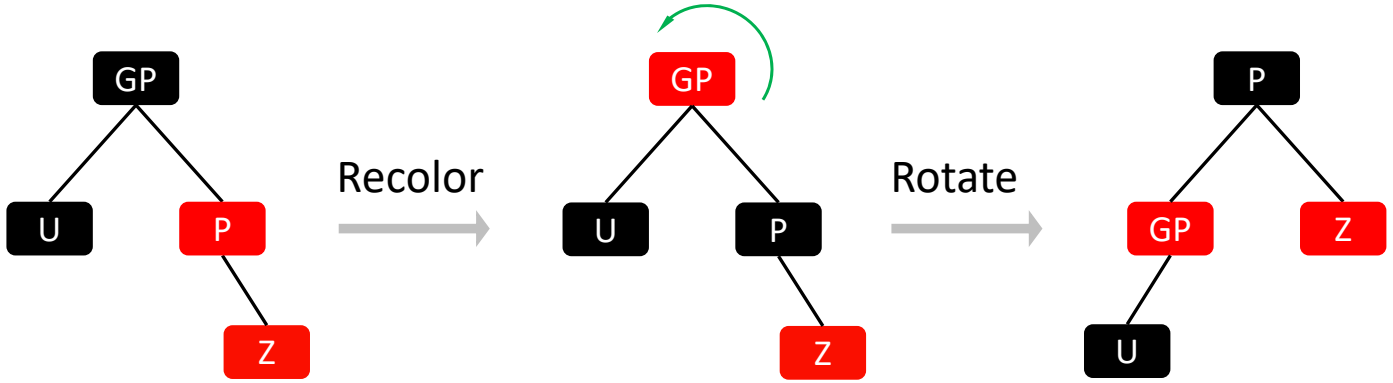
- Solution:
  - Objective:
    - Make the triangle GP-P-Z a straight line
  - Rotate parent
    - If z is the left child, rotate right
    - If z is the right child, rotate left

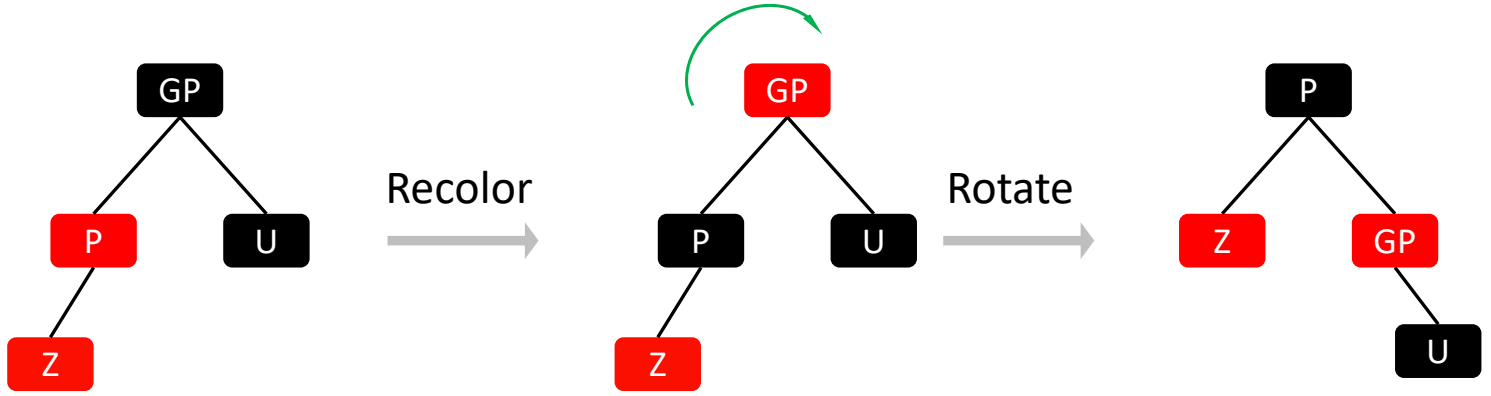




Case 3: Z has a black uncle and GP-P-Z is a straight line

- Objective:
  - Make the tree more balance
- Solution:
  - Recolor P to P
  - Recolor GP to GP
  - Rotate grandpa
    - If z is the right child, rotate left
    - If z is the left child, rotate right





Fix-up algorithm

```
RB-INSERT-FIXUP(T, z)
 1  while z.p.color == RED
 2      if z.p == z.p.p.left
 3          y = z.p.p.right
 4          if y.color == RED
 5              z.p.color = BLACK                          // case 1
 6              y.color = BLACK                            // case 1
 7              z.p.p.color = RED                          // case 1
 8              z = z.p.p                                  // case 1
 9          else if z == z.p.right
10                  z = z.p                                // case 2
11                  LEFT-ROTATE(T, z)                      // case 2
12              z.p.color = BLACK                          // case 3
13              z.p.p.color = RED                          // case 3
14              RIGHT-ROTATE(T, z.p.p)                     // case 3
15      else (same as then clause
                with "right" and "left" exchanged)
16  T.root.color = BLACK
```