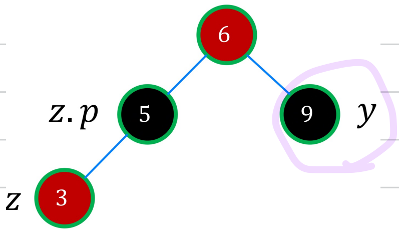
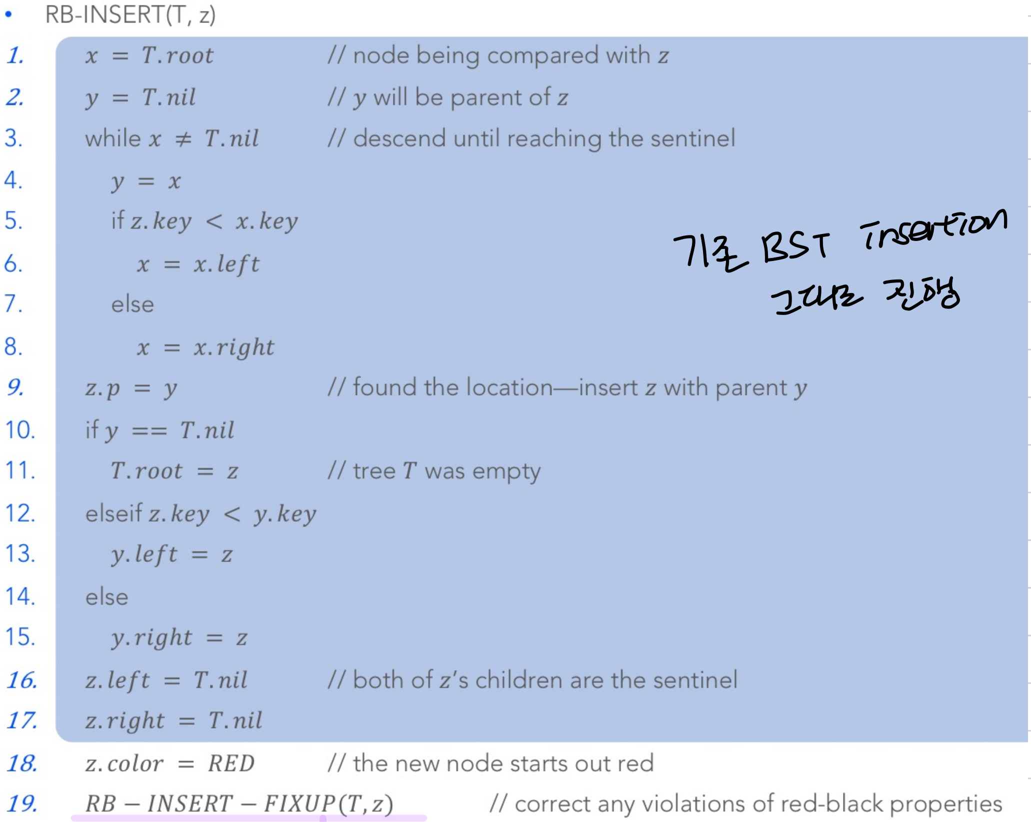
RBT Insertion



⇒ RBT로 고칠때 봐야 하는 것이 새로 넣은 node의 "uncle"을 봐야 함! (= parent의 sibling)

↳ uncle에 따라 case가 나뉘짐



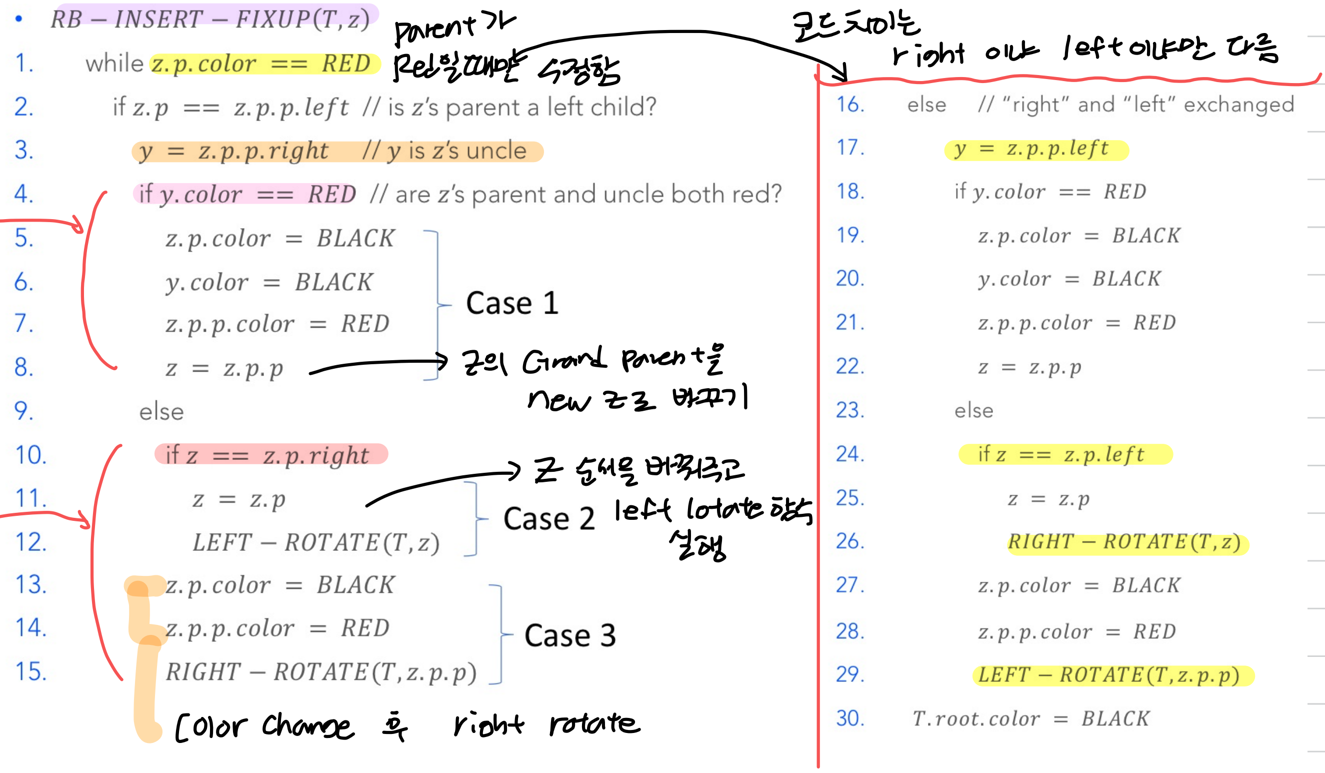
- RB-INSERT(T, z)

```
x = T.root                // node being compared with z
y = T.nil                 // y will be parent of z
while x ≠ T.nil           // descend until reaching the sentinel
    y = x
    if z.key < x.key
        x = x.left
    else
        x = x.right
z.p = y                   // found the location—insert z with parent y
if y == T.nil
    T.root = z            // tree T was empty
elseif z.key < y.key
    y.left = z
else
    y.right = z
z.left = T.nil            // both of z's children are the sentinel
z.right = T.nil
z.color = RED             // the new node starts out red
RB-INSERT-FIXUP(T, z)     // correct any violations of red-black properties
```

기존 BST insertion 그대로 진행

새로 넣은 z의 color는 무조건 Red로 들어감

↳ RBT properties가 어겨지는 부분들을 고치기 시작



- RB-INSERT-FIXUP(T, z)

```
while z.p.color == RED
    if z.p == z.p.p.left          // is z's parent a left child?
        y = z.p.p.right           // y is z's uncle
        if y.color == RED         // are z's parent and uncle both red?
            z.p.color = BLACK        ┐
            y.color = BLACK          │ Case 1
            z.p.p.color = RED        │
            z = z.p.p                ┘
        else
            if z == z.p.right
                z = z.p              ┐ Case 2
                LEFT-ROTATE(T, z)    ┘
            z.p.color = BLACK        ┐
            z.p.p.color = RED        │ Case 3
            RIGHT-ROTATE(T, z.p.p)   ┘
    else                          // "right" and "left" exchanged
        y = z.p.p.left
        if y.color == RED
            z.p.color = BLACK
            y.color = BLACK
            z.p.p.color = RED
            z = z.p.p
        else
            if z == z.p.left
                z = z.p
                RIGHT-ROTATE(T, z)
            z.p.color = BLACK
            z.p.p.color = RED
            LEFT-ROTATE(T, z.p.p)
T.root.color = BLACK
```

parent가 Red일때만 수정함

코드차이는 right 이냐 left이냐만 다름

uncle이 red인 경우

z의 Grand parent를 new z로 바꾸기

uncle이 Black인 경우

z 순서를 바꿔주고 left lotate 한번 실행

Color change 후 right rotate

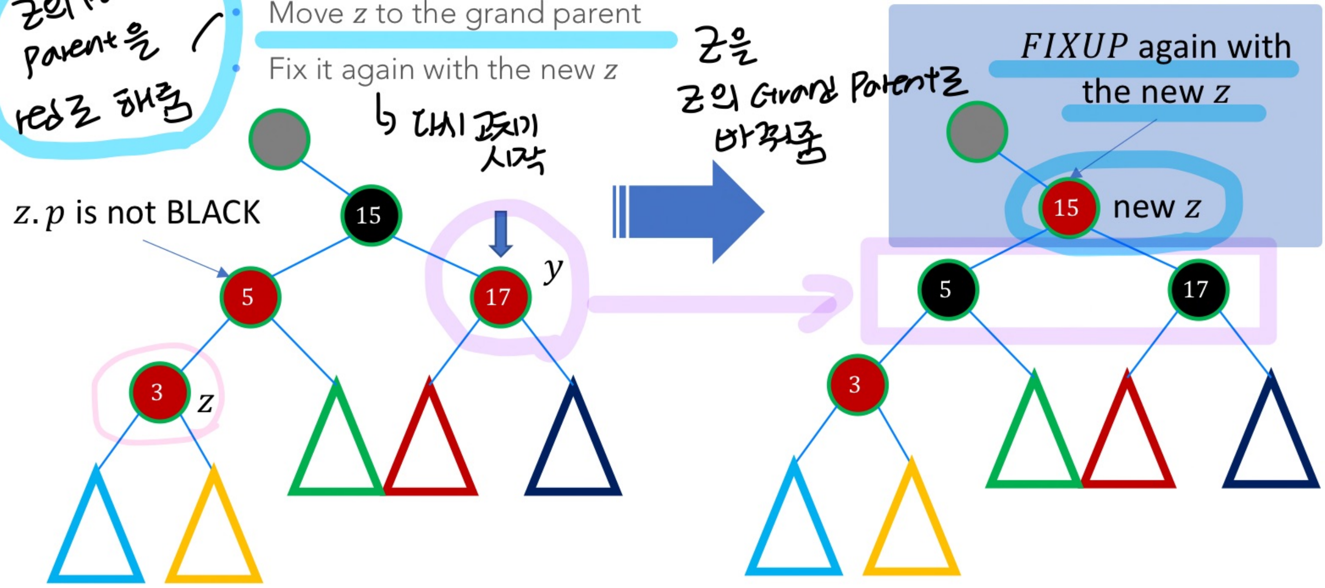
Case ①

- $RB - INSERT - FIXUP(T,z)$
  - Case 1: RED uncle y
    - Recolor → Parent 와 uncle을 다 black으로 바꿔줌
    - Move z to the grand parent
    - Fix it again with the new z

z의 Parent의 parent을 red로 해줌

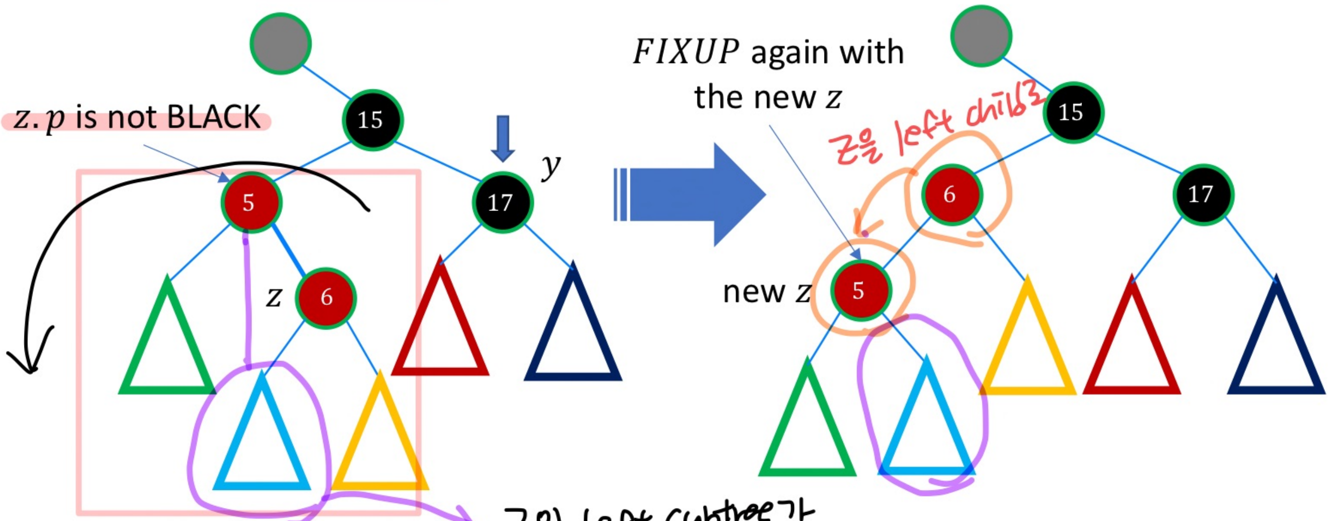
z을 z의 Grand Parent로 바꿔줌

↳ 다시 고치기 시작



Case ②

- $RB - INSERT - FIXUP(T,z)$
  - Case 2: BLACK uncle y and z is a right child
    - Left rotate around the parent then you have Case 3



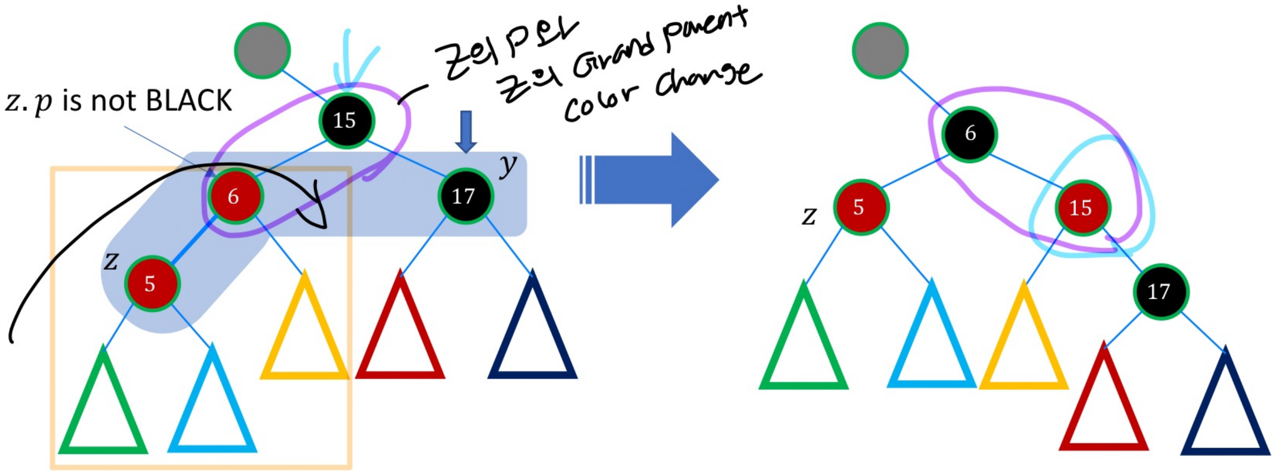
left rotate

z의 left subtree가 5의 right subtree로 바뀜

become Case 3

Case ③

- $RB - INSERT - FIXUP(T, z)$
  - Case 3: BLACK uncle y and z is a left child
    - Right rotate around the grand parent and recolor
    - We have an RBT



Insertion time 속도

- The height of a red-black tree on n nodes is $O(\lg n)$
- Lines 1–18 of $RB - INSERT$ take $O(\lg n)$ time
- The running time of $RB - INSERT - FIXUP$ is $O(\lg n)$
  - The while loop repeats only if case 1 occurs,

    and then the pointer z moves two levels up the tree
    - The total number of times the while loop can be executed is $O(\lg n)$
    - The while loop never performs more than two rotations
      - The while loop terminates if case 2 or case 3 is executed
- Thus, $RB - INSERT$ takes a total of $O(\lg n)$ time

BST ⇒ $O(h)$

RBT ⇒ $h \leq 2\log(n+1)$

$= O(\log n)$

$z = z.pp$