효율적인 이진 탐색 트리

- **AVL 트리**
- **Red-black 트리**

Binary Search Tree (BST: 이진 탐색 트리)

◆ **Binary Search Tree의 기본적인 연산 수행시간**

	원소 검색	원소 삽입	원소 삭제	
수행 시간	O(h)	O(h)	O(h)	h: BST 높이
평균적인 수행시간	O(log n)	O(log n)	O(log n)	n: 노드 수

1. AVL 트리

- **AVL(Adelson-Velskii and Landis) trees are height-balanced binary search trees**

- Adelson-Velskii and E.M. Landis가 제안

- **균형인수(balance factor)**
  - 노드 X의 균형인수 BF(X) = $h_L - h_R$

    h_L은 X의 왼쪽 서브트리의 높이

    h_R은 X의 오른쪽 서브트리의 높이

- **AVL 트리: 모든 노드 X에 대해서 BF(X)= -1, 0, 1**

  AVL 트리의 높이: O(log n)

탐색은 $O(\log_2 n)$

AVL 트리 – 높이 분석 1

- 노드수가 n인 AVL tree의 최대 높이분석
  - **N(h) = minimum number of nodes in an AVL tree of height h.**
  - **Basis**
    - N(1) = 1, N(2) = 2
  - **Induction**
    - N(h) = N(h-1) + N(h-2) + 1
  - **Analysis**

N(h-1) > N(h-2)이다.

따라서, $N(h) > 2N(h-2)$

$N(h) > 2^2 N(h-4)$

…

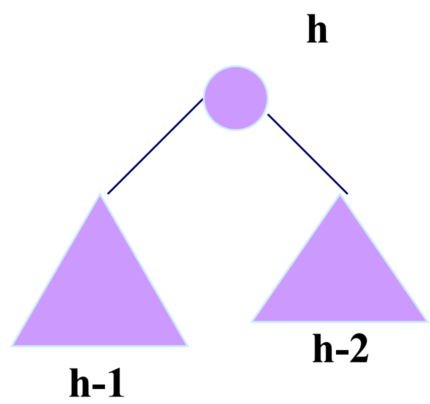
$n \geq N(h) > 2^{h/2}$ → 양변에 ln 취함.

$h/2 < \log_2 n$

$h < 2 \log_2 n$ $\log_2 n$에 두배를 넘지 않음.

$N(h) = N(h-1) + N(h-2) + 1$

h : O(log n)

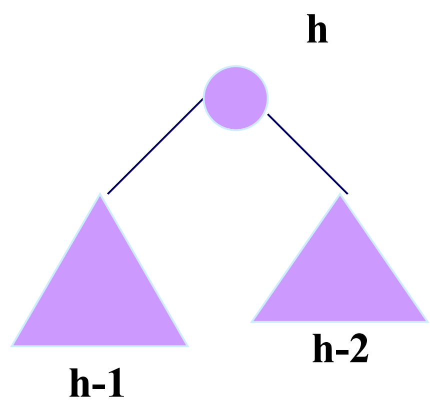


AVL 트리 – 높이 분석 2

- **N(h) = minimum number of nodes in an AVL tree of height h.**
- **Analysis (recall Fibonacci analysis)**
  - $N(h) + 1 = N(h-1) + 1 + N(h-2) + 1$
  - Let $F(h) = N(h) + 1$
  - $F(h) = F(h-1) + F(h-2), h >= 2$

    $1 \qquad , h = 1$

    $2 \qquad , h = 2$
  - $N(h) \geq \phi^h - 1 \quad (\phi \approx 1.62)$



AVL 트리 – 높이 분석 2

- **$N(h) \geq \phi^h - 1$ ($\phi \approx 1.62$)**
- **Suppose we have n nodes in an AVL tree of height h.**
  - $n \geq N(h)$ (because N(h) was the minimum)
  - $n \geq \phi^h - 1$, hence $\log_\phi (n+1) \geq h$ (relatively well balanced tree!!)
  - $h \leq 1.44 \log_2 n + 1$(i.e., h: $O(\log n)$)

AVL 트리 원소 삽입

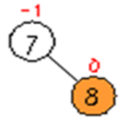
+1 -1 0은 해당안됨

- Y: 이진탐색트에서 삽입된 노드
- 삽입된 노드 Y에 가장 가까우면서 균형인수가 ±2인 조상 노드 A에 대하여, 다음 4가지 경우로 나누어 회전
  - LL: 새 노드 Y는 A의 왼쪽 서브트리의 왼쪽 서브트리에 삽입
  - LR: Y는 A의 왼쪽 서브트리의 오른쪽 서브트리에 삽입
  - RR: Y는 A의 오른쪽 서브트리의 오른쪽 서브트리에 삽입
  - RL: Y는 A의 오른쪽 서브트리의 왼쪽 서비트리에 삽입
- 단일회전(Single rotation) : LL과 RR의 경우 불균형을 바로잡는 변환
- 이중회전(Double rotation) : LR과 RL의 경우 불균형을 바로잡는 변환

예

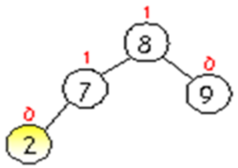
7, 8, 9, 2, 1, 5, 3, 6, 4

(a) 7 삽입

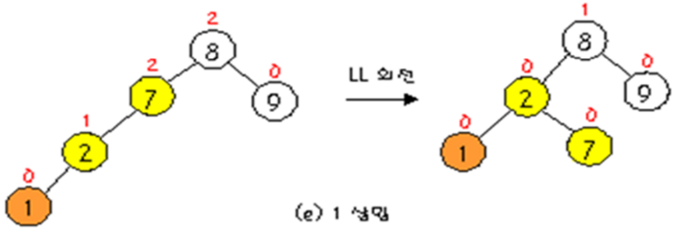


(b) 8 삽입

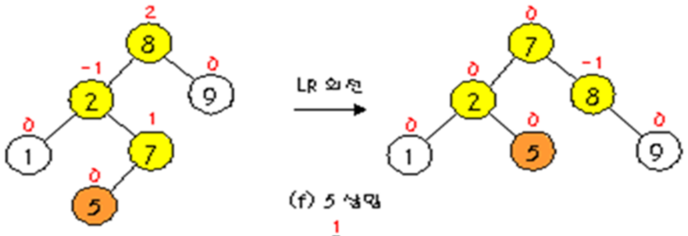
(c) 9 삽입



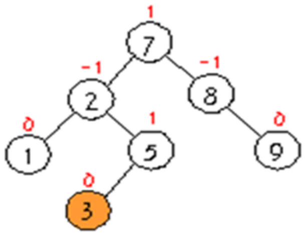
(d) 2 삽입



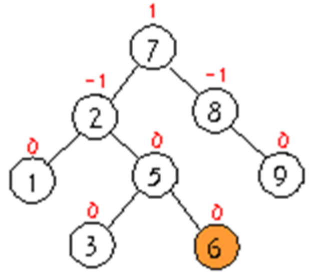
(e) 1 삽입



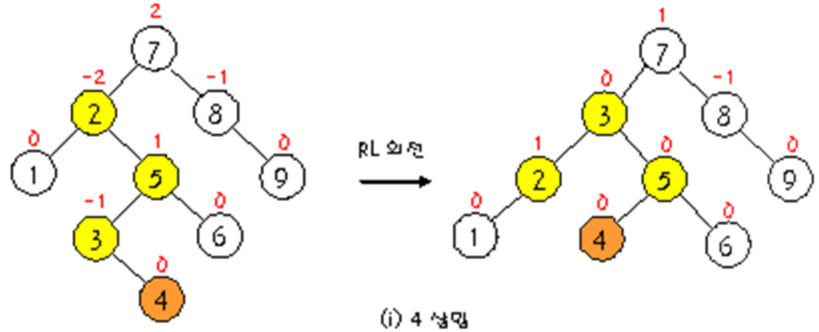
(f) 5 삽입



(g) 3 삽입



(h) 6 삽입



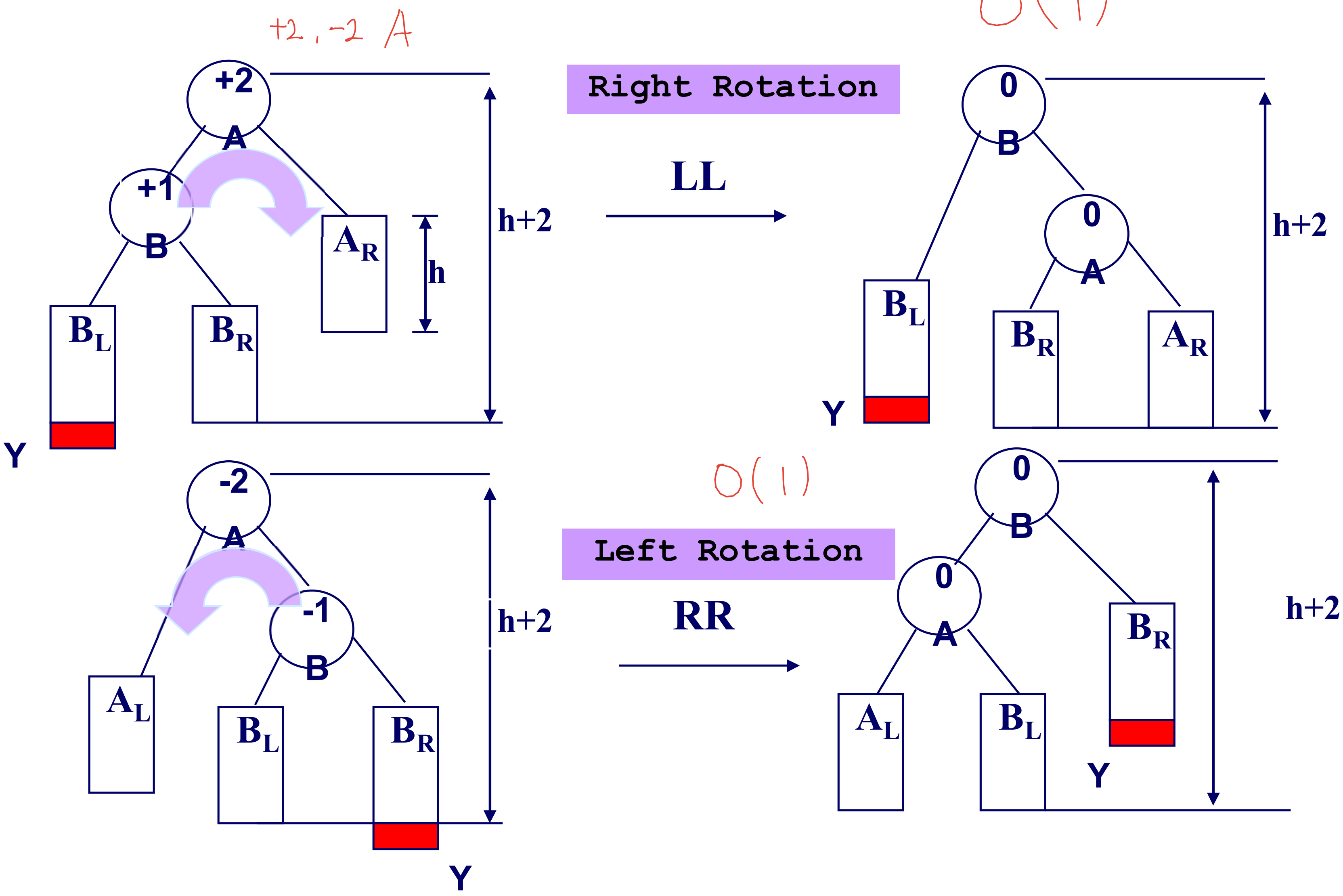
(i) 4 삽입

회전방법

- **LL: A**부터 **Y**까지의 경로상의 노드들을 오른쪽으로 회전시킨다**.**
- **LR: A**부터 **Y**까지의 경로상의 노드들을 왼쪽**-**오른쪽으로 회전시킨다**.**
- **RR: A**부터 **Y**까지의 경로상의 노드들을 왼쪽으로 회전시킨다**.**
- **RL: A**부터 **Y**까지의 경로상의 노드들을 오른쪽**-**왼쪽으로 회전시킨다**.**

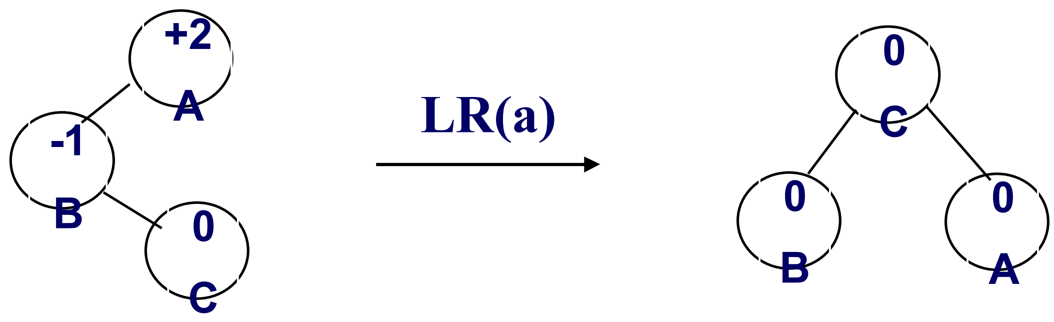
LL 및 RR

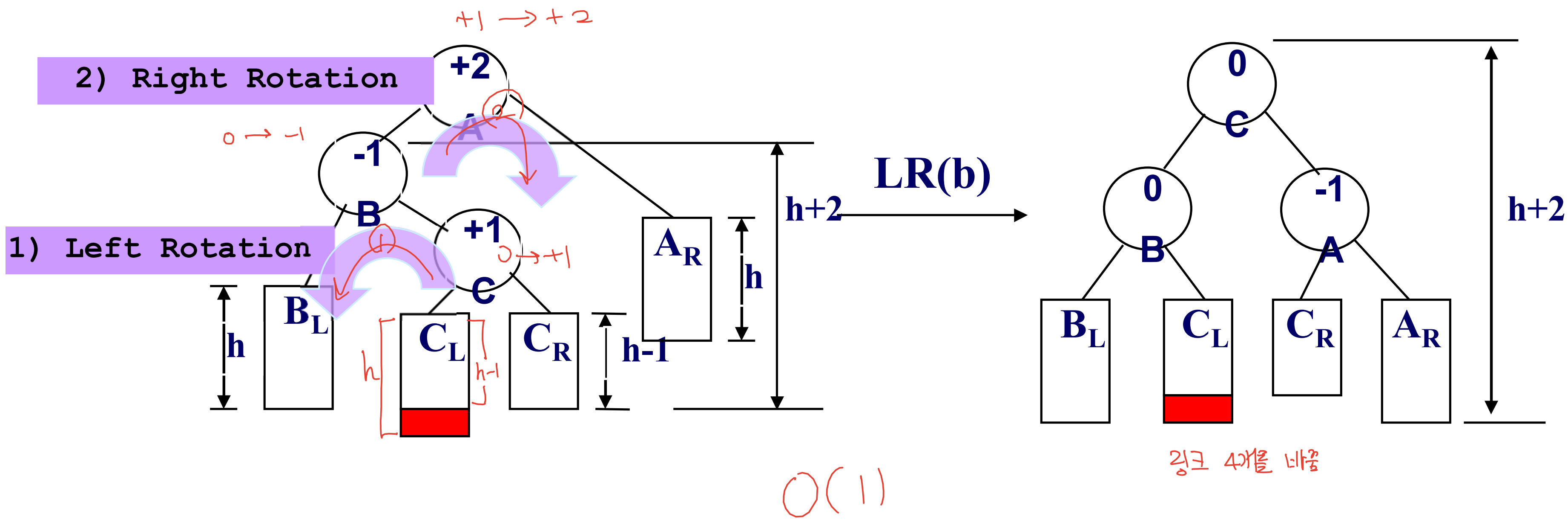
- LL 및 RR


+2 , -2 A
O(1)
+2
A
+1
B
A_R
h
B_L
B_R
Y
h+2
Right Rotation
LL
0
B
0
A
B_L
B_R
A_R
Y
h+2
-2
A
-1
B
A_L
B_L
B_R
Y
h+2
O(1)
Left Rotation
RR
0
B
0
A
A_L
B_L
B_R
Y
h+2

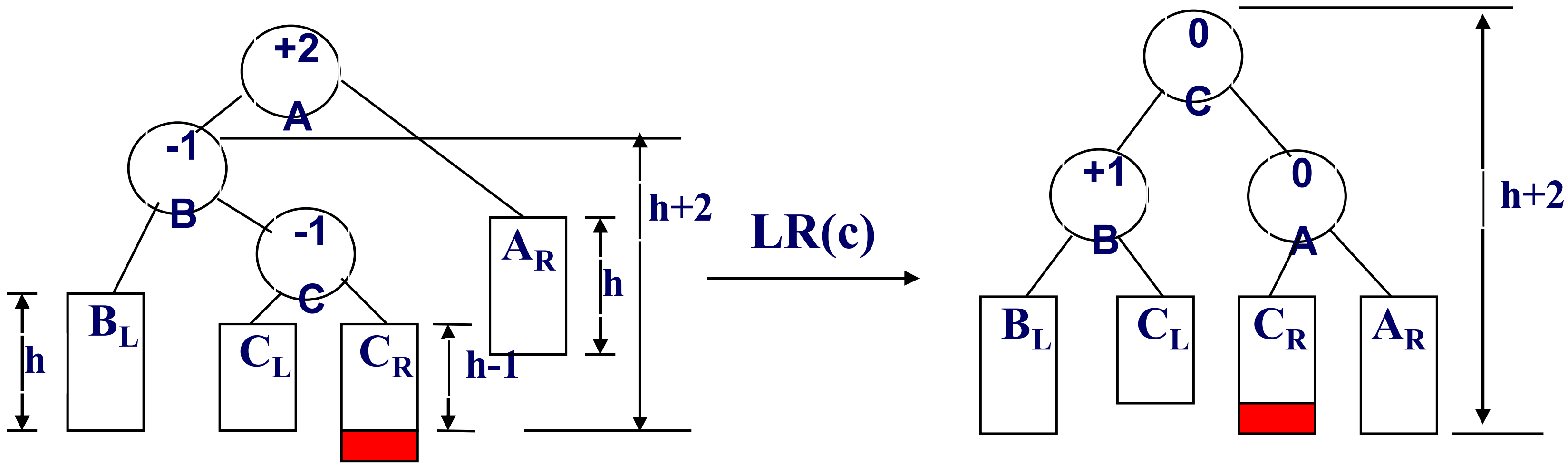
LR

◆ LR: left rotation과 right rotation을 수행



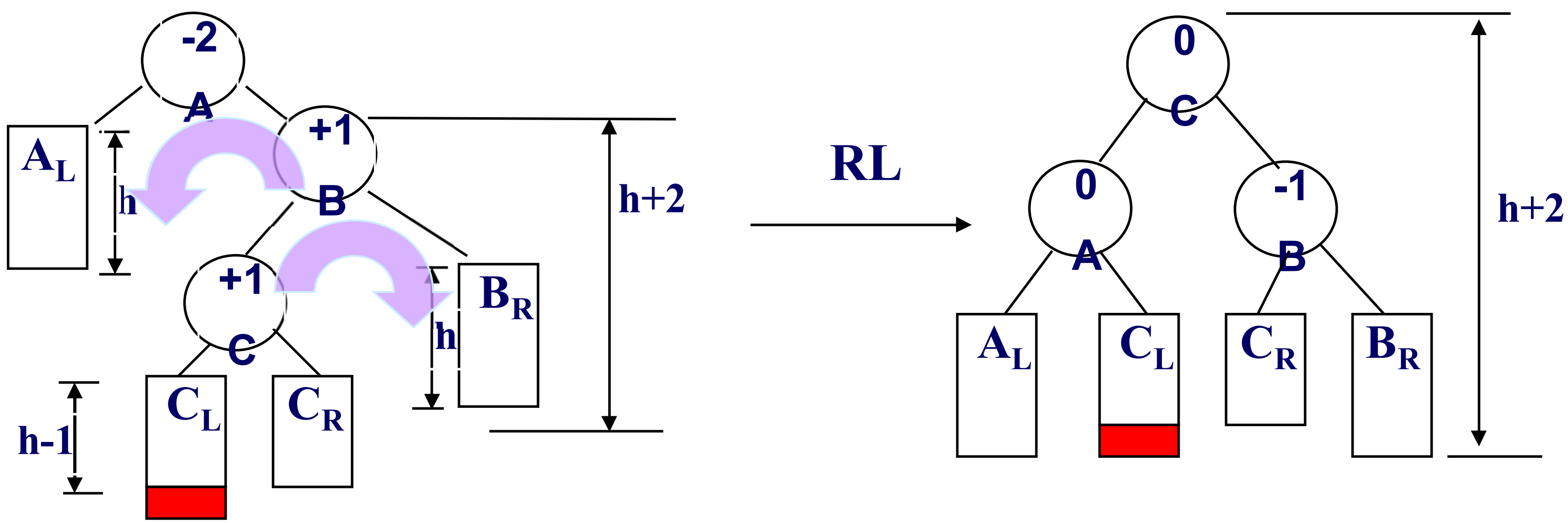


LR



RL

- RL: LR과 대칭적으로 해결
- RL: right rotation과 left rotation을 수행



AVL 트리의 수행시간

삭제는 pass

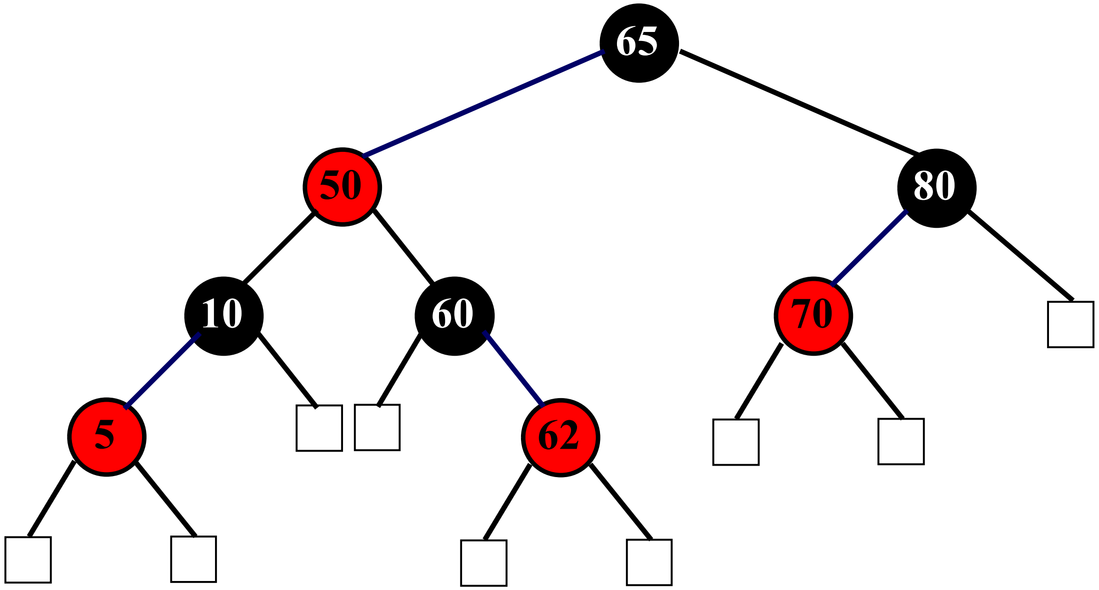
	원소 검색	원소 삽입	원소 삭제	
수행 시간	O(log n)	O(log n)	O(log n)	n: 노드 수

2. 레드-블랙(red-black) 트리

◆ 정의
다음 성질을 가지는 이진탐색트리이다.

- 1. 모든 노드는 red 혹은 black 색을 가진다.
- 2. 루트의 색은 블랙이다.
- 3. 레드 색을 가지는 노드의 자식노드가 있으면, 이 자식노드 색은 블랙이다.
- 4. 루트에서 모든 null child 까지의 경로들에 있는 블랙 색 노드의 수는 동일하다.

레드-블랙 트리



□ NULL

높이 $O(\log_2 n)$

레드-블랙 트리

시험범위 X

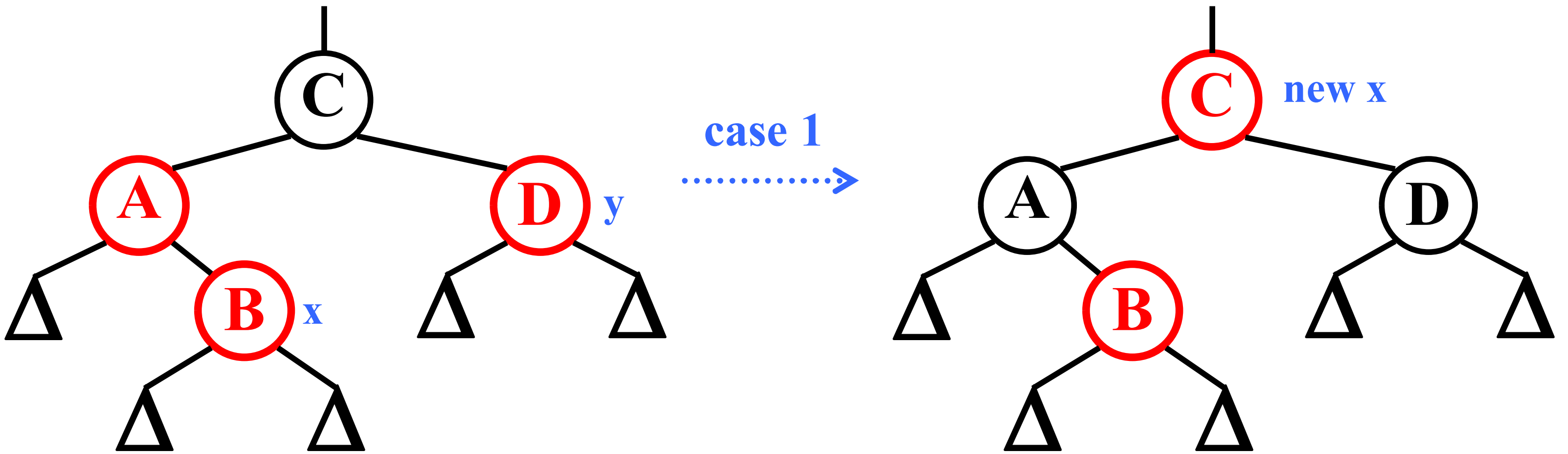
```
rbInsert(x) // red-black tree insertion
  treeInsert(x);                  // x는 삽입되는 노드
  x->color = RED;
  // Move violation of #3 up tree, maintaining #4 as invariant:
  while (x!=root && x->p->color == RED) // color는 노드의 색임
      if (x->p == x->p->p->left)  // p는 parent를 나타내고 left는 left child를,
          y = x->p->p->right           // right는 right child를 나타낸다
          if (y->color == RED)
              x->p->color = BLACK
              y->color = BLACK
              x->p->p->color = RED                    Case 1
              x = x->p->p
          else   // y->color == BLACK
              if (x == x->p->right)
                  x = x->p
                  leftRotate(x)                       Case 2
              x->p->color = BLACK
              x->p->p->color = RED
              rightRotate(x->p->p)
      else    // x->p == x->p->p->right               Case 3
      (same as above, but with
       "right" & "left" exchanged)

                                                      Case 4-6
```

RB Insert: Case 1

```
if (y->color == RED)
    x->p->color = BLACK;
    y->color = BLACK;
    x->p->p->color = RED;
    x = x->p->p;
```

- Case 1: “uncle” (노드 y) is red
- In figures below, all Δ’s are equal-black-height subtrees

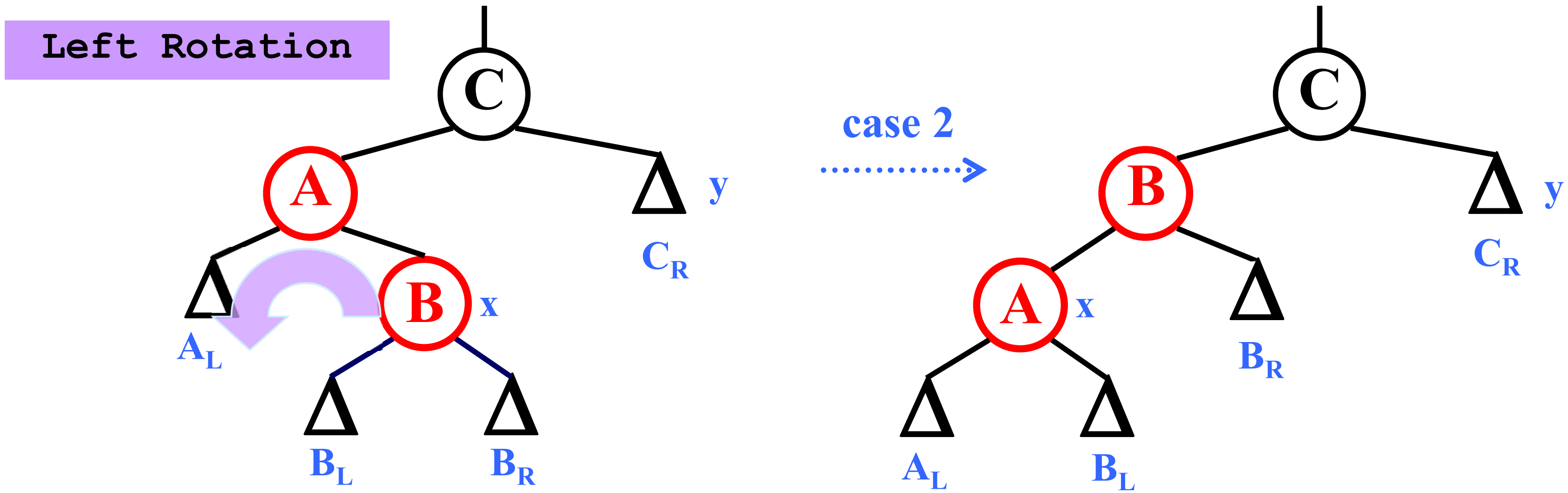


Same action whether x is a left or a right child

RB Insert: Case 2

```
if (x == x->p->right)
     x = x->p;
     leftRotate(x);
// continue with case 3 code
```

- **Case 2:**
  - "Uncle" is black
  - Node *x* is a right child
- **Transform to case 3 via a left-rotation**

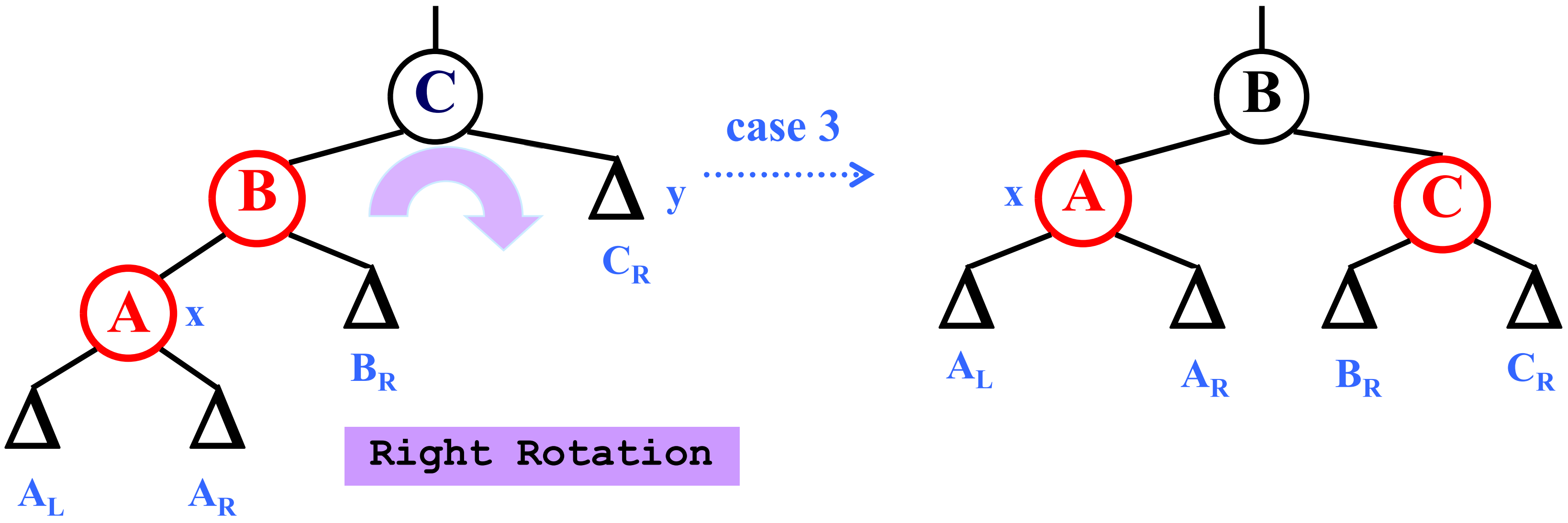


Transform case 2 into case 3 (x is left child) with a left rotation

RB Insert: Case 3

```
x->p->color = BLACK;
x->p->p->color = RED;
rightRotate(x->p->p);
```

- **Case 3:**
  - "Uncle" is black
  - Node *x* is a left child
- **Change colors; rotate right**

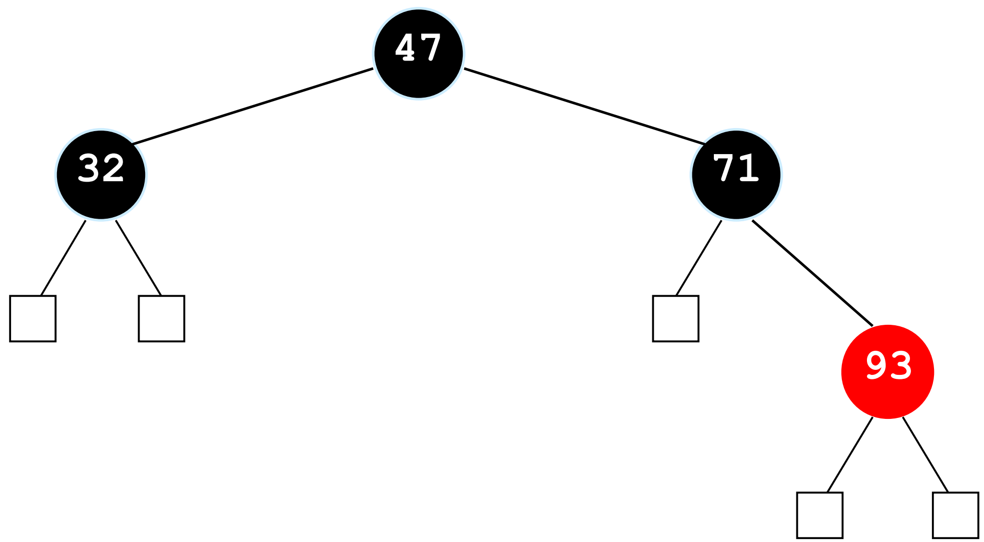


Perform some color changes and do a right rotation

RB Insert: Cases 4-6

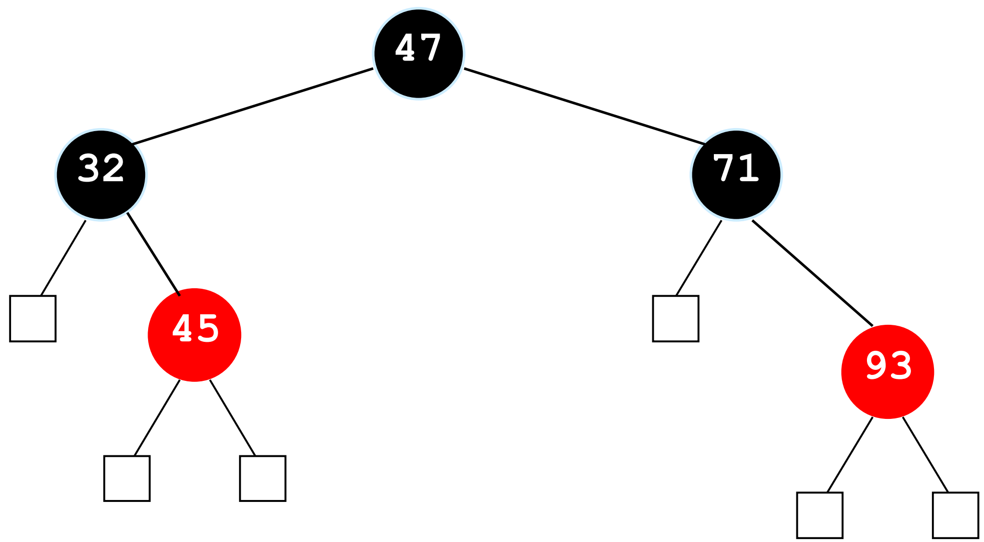
- Cases 1-3 hold if *x*'s parent is a left child
- If *x*'s parent is a right child, cases 4-6 are symmetric (swap left for right)

Insertion Example



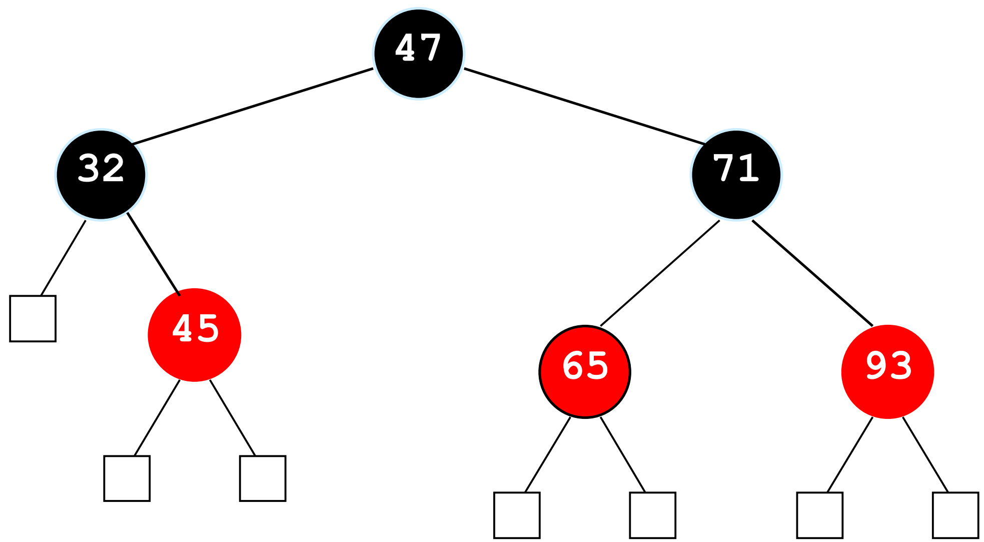
Insertion Example

Insert 45



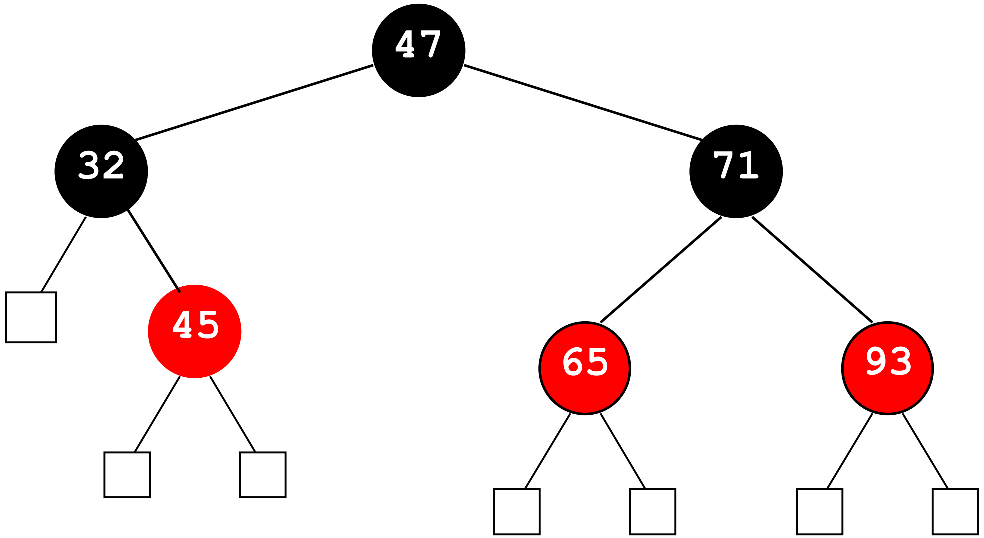
Insertion Example

Insert 65



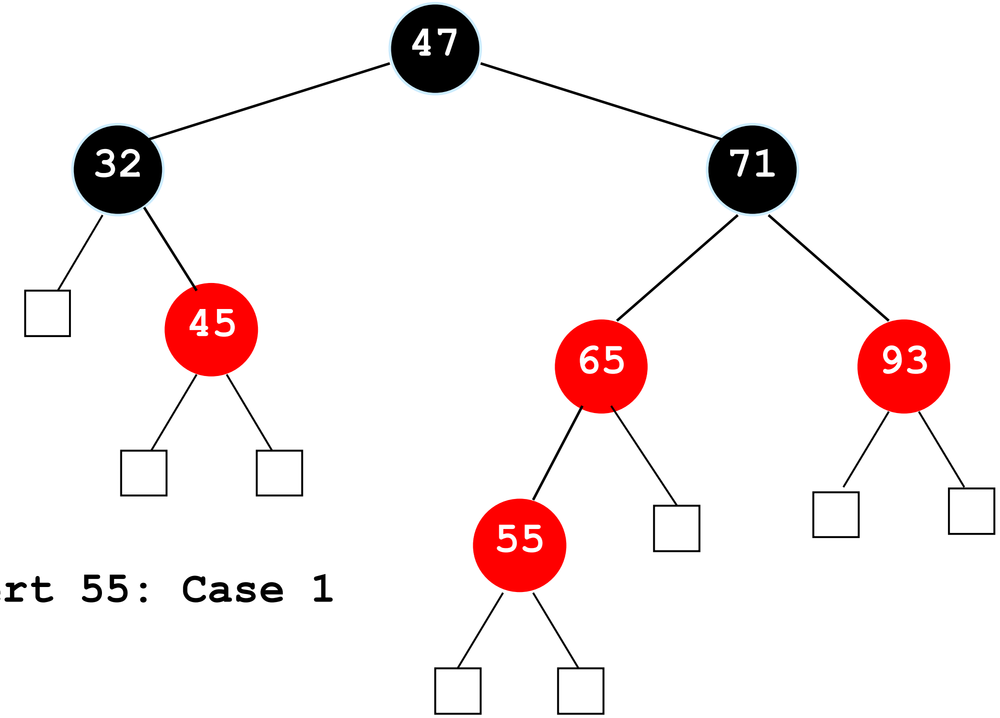
Insertion Example

Insert 55



Insertion Example

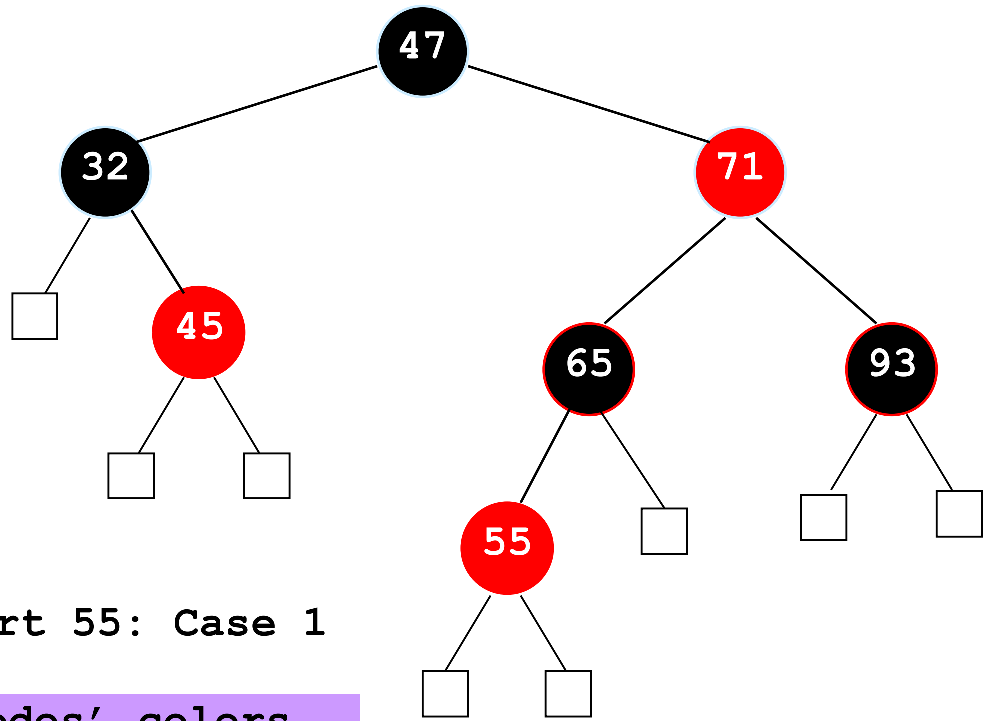
Insert 55



Insert 55: Case 1

Insertion Example

Insert 55

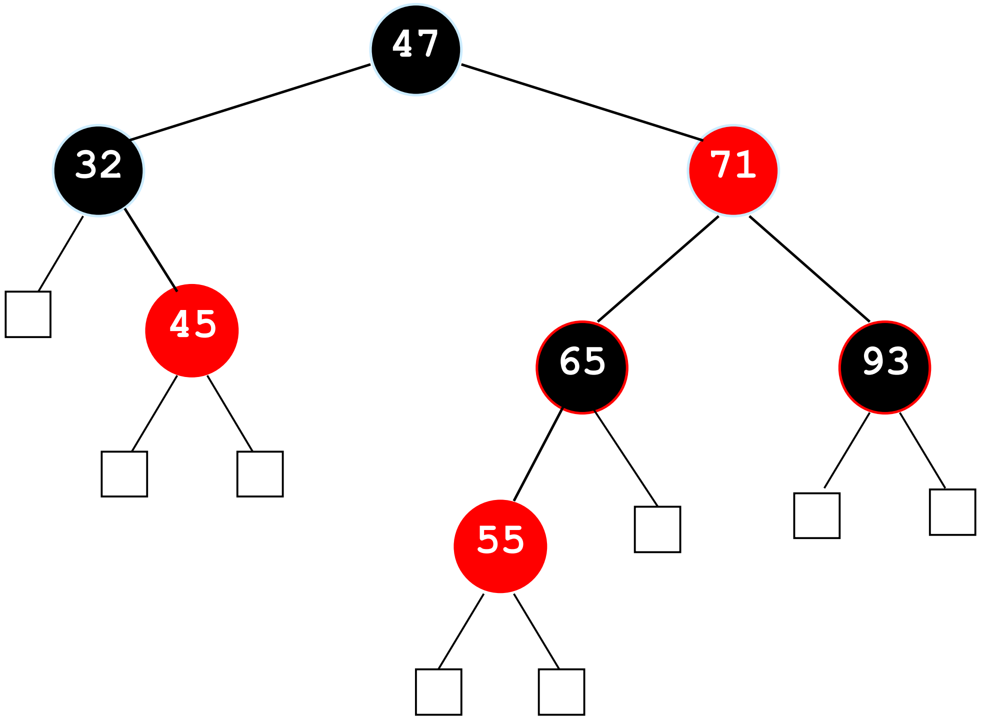


Insert 55: Case 1

change nodes' colors

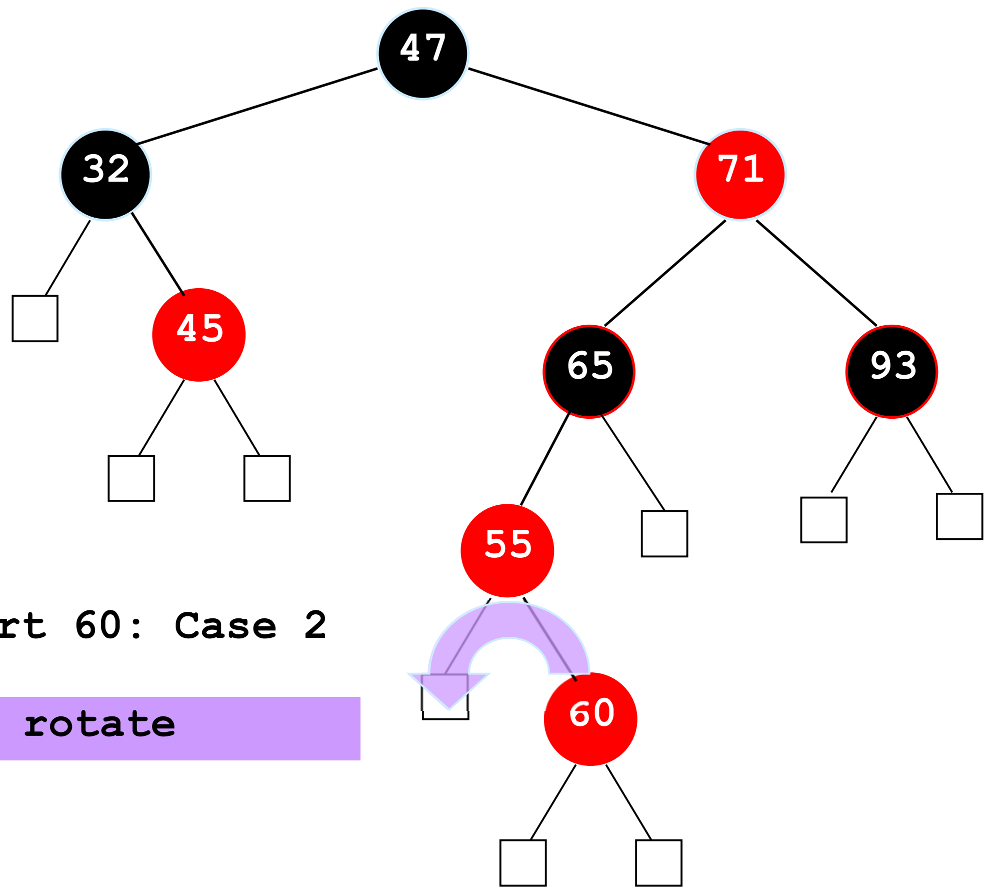
Insertion Example

Insert 60



Insertion Example

Insert 60

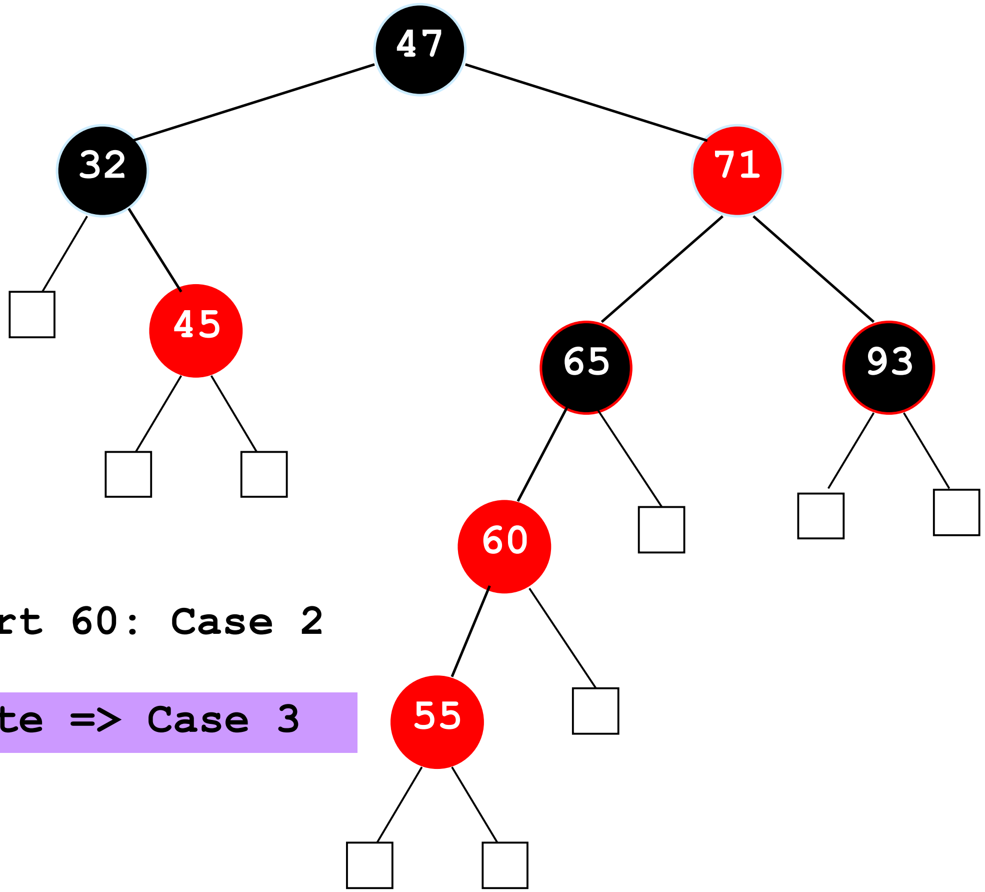


Insert 60: Case 2

Left rotate

Insertion Example

Insert 60

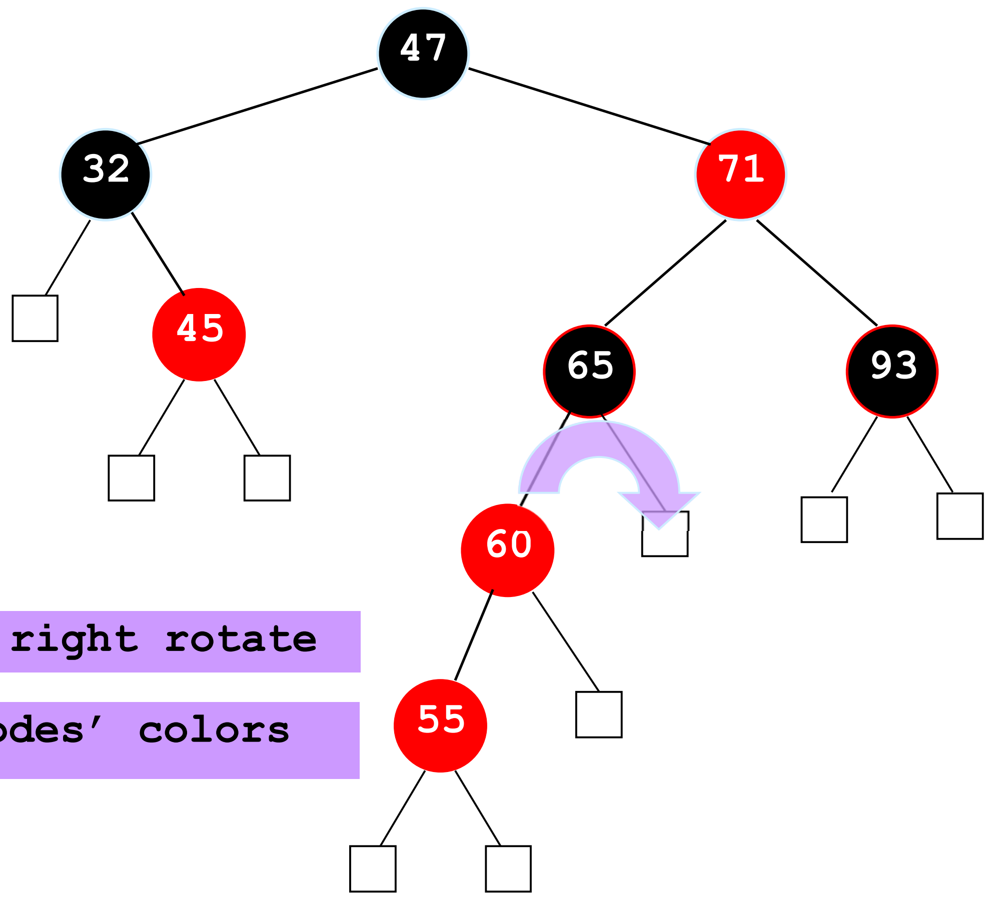


Insert 60: Case 2

Left rotate => Case 3

Insertion Example

Insert 60

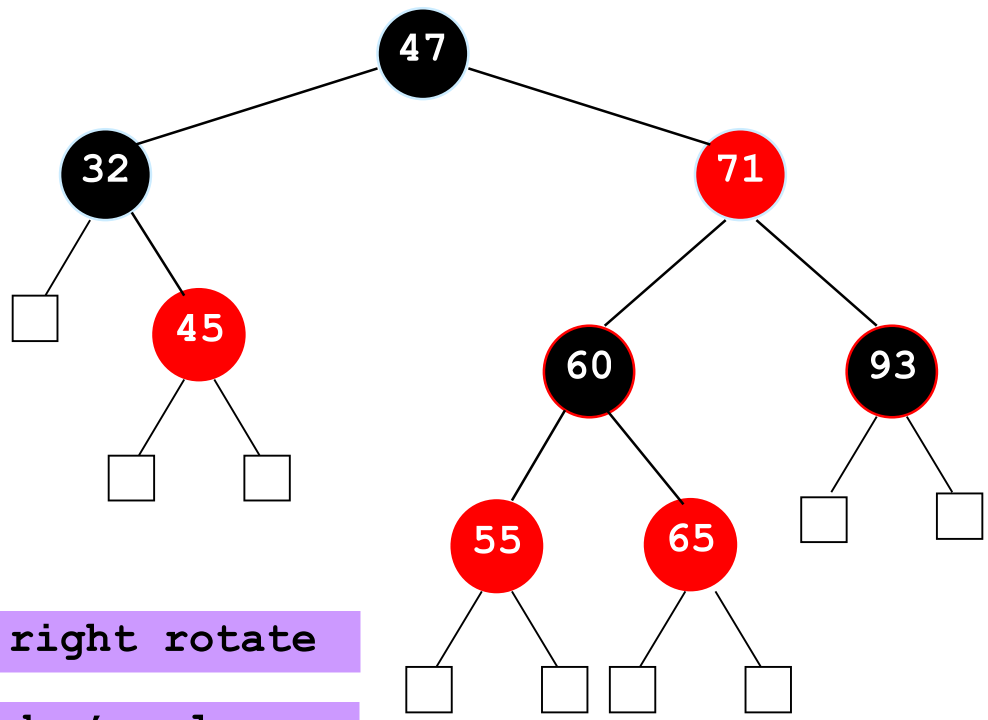


Case 3 => right rotate

change nodes' colors

Insertion Example

Insert 60

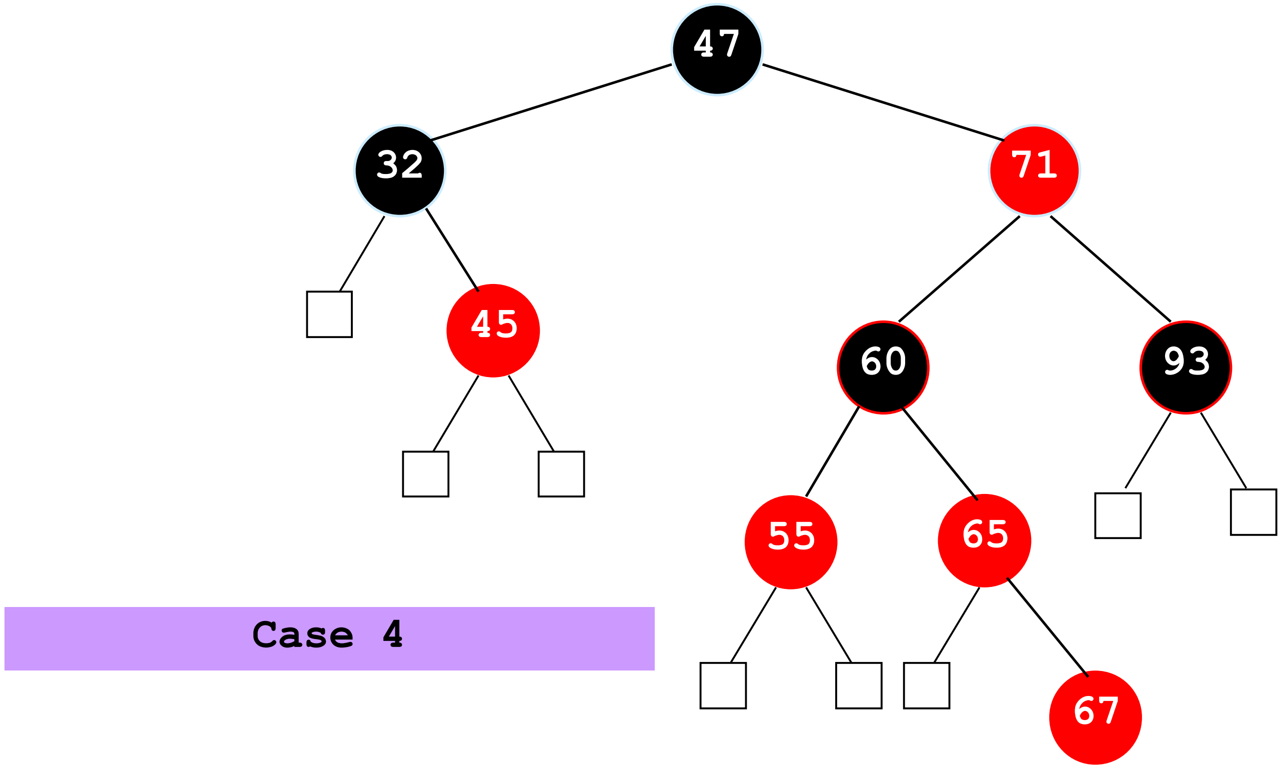


Case 3 => right rotate

change nodes' colors

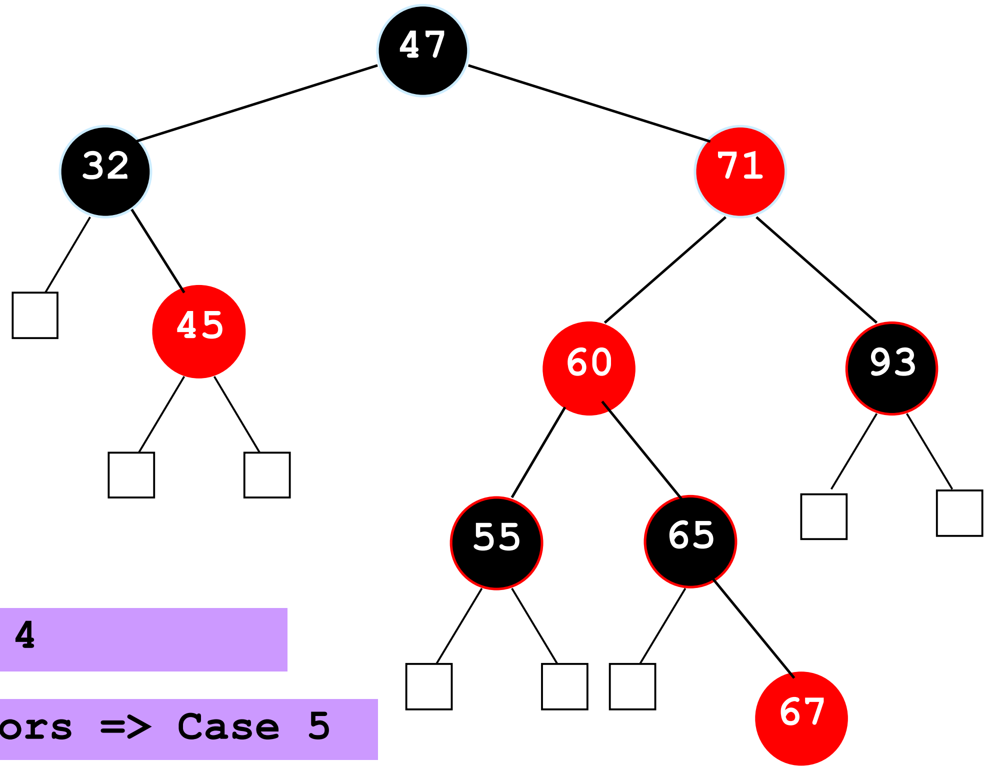
Insertion Example

Insert 67



Insertion Example

Insert 67

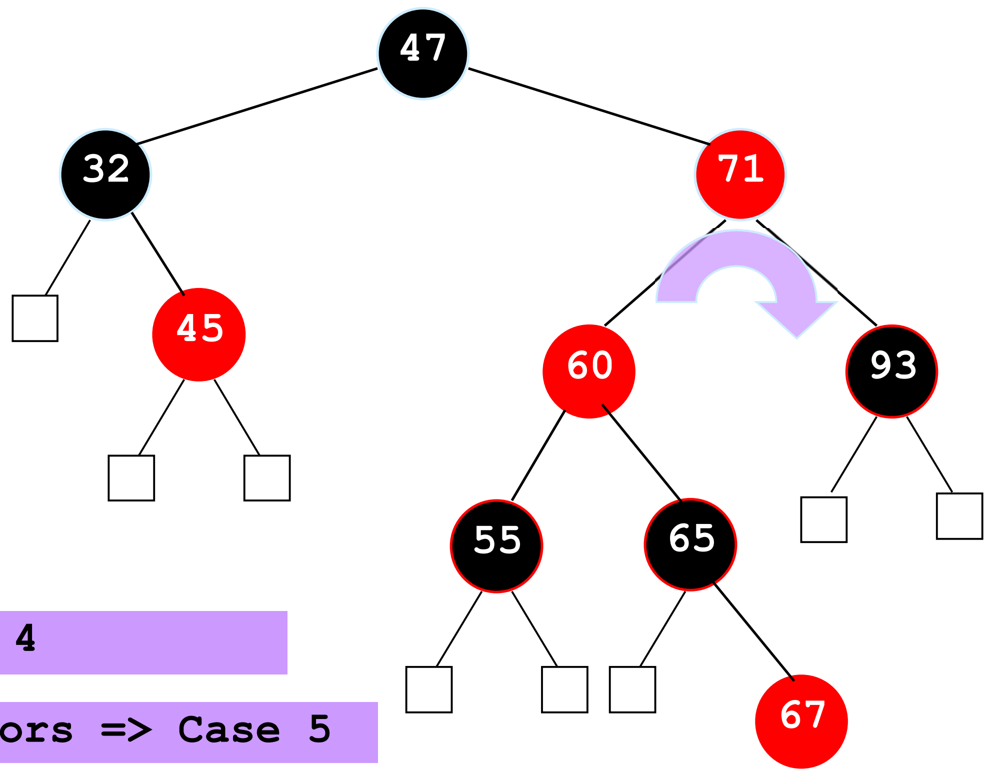


Case 4

change nodes' colors => Case 5

Insertion Example

Insert 67

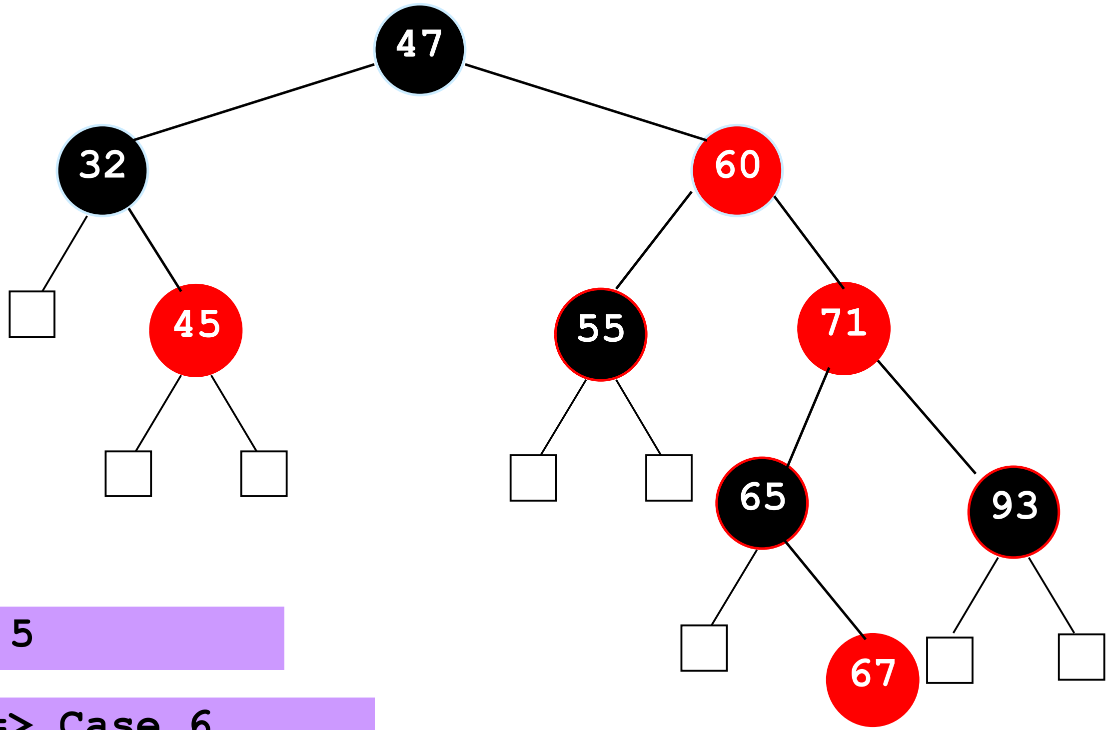


Case 4

change nodes' colors => Case 5

Insertion Example

Insert 67

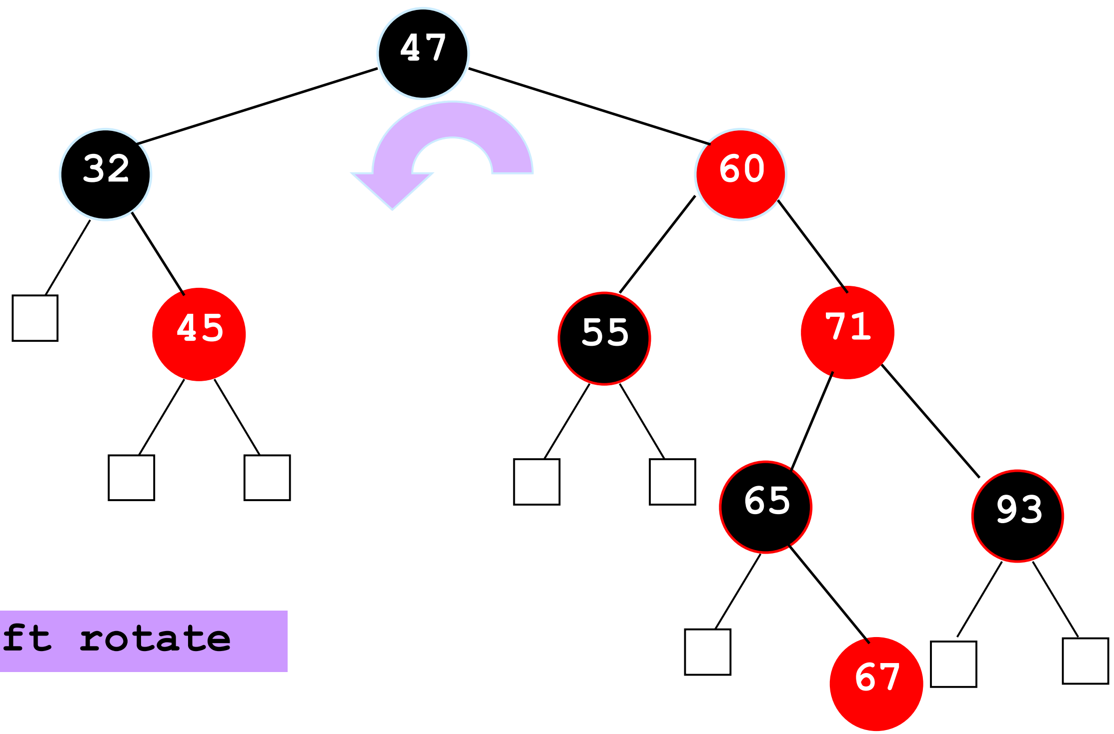


Case 5

Left rotate => Case 6

Insertion Example

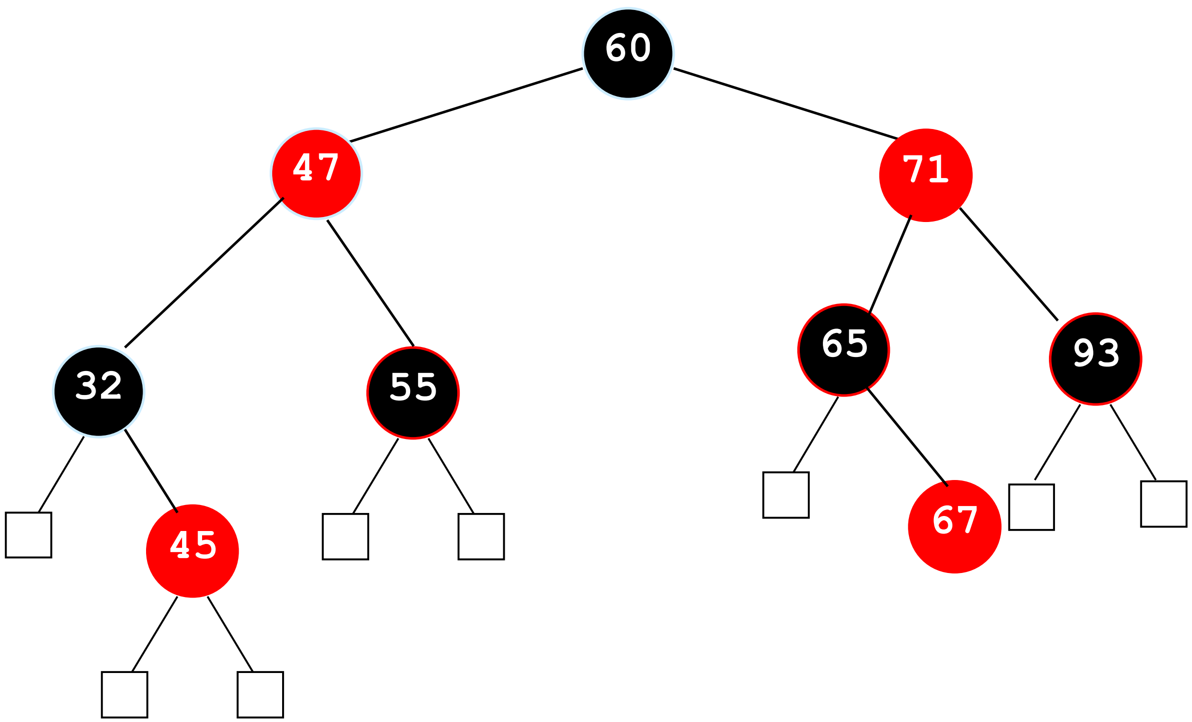
Insert 67



Case 6 => Left rotate

Insertion Example

Insert 67



Case 6 => Left rotate

change nodes' colors

레드-블랙 트리

- 보조 정리
  - 레드-블랙 트리 높이 h, 트리 내부 노드수 n이면 $h \le 2 \log_2(n+1)$

- 수행시간

	원소 검색	원소 삽입	원소 삭제	
수행 시간	O(log n)	O(log n)	O(log n)	n: 노드 수

- **STL map의 자료구조는 red-black tree를 사용한다.**