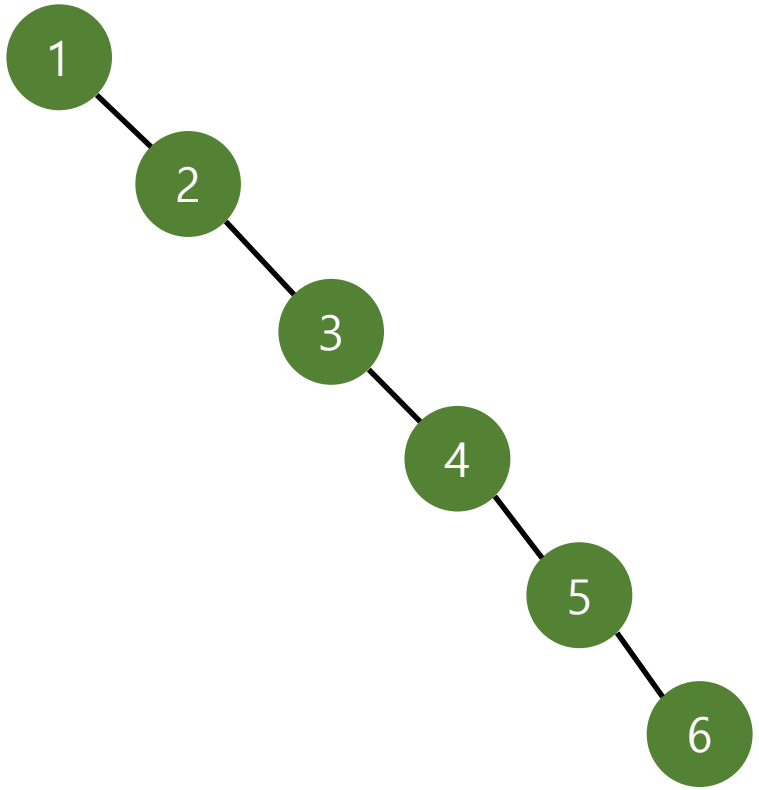
Red-black Tree

왜 균형 이진 트리(balanced binary tree)인가?

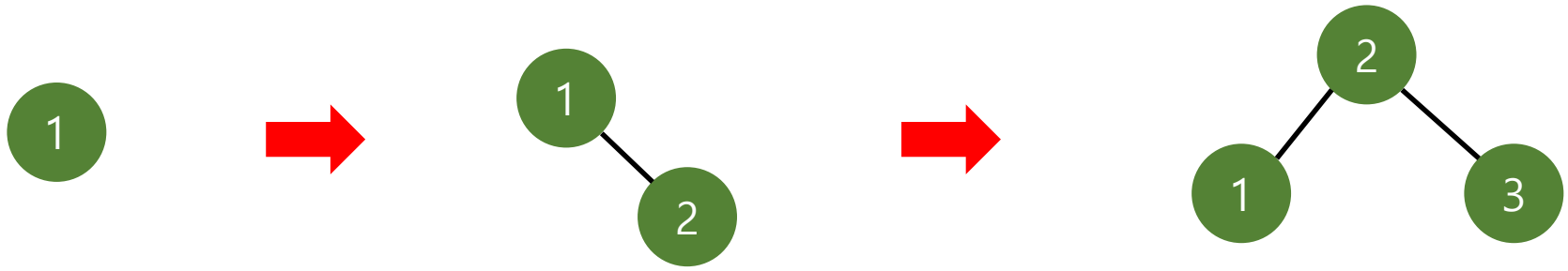
BST에서 최악의 경우
search의 빅오는

O(n)



왜 균형 이진 트리(balanced binary tree)인가?

노드가 insert 되거나 delete 되면
알아서 균형을 맞춰줄 수는 없을까?

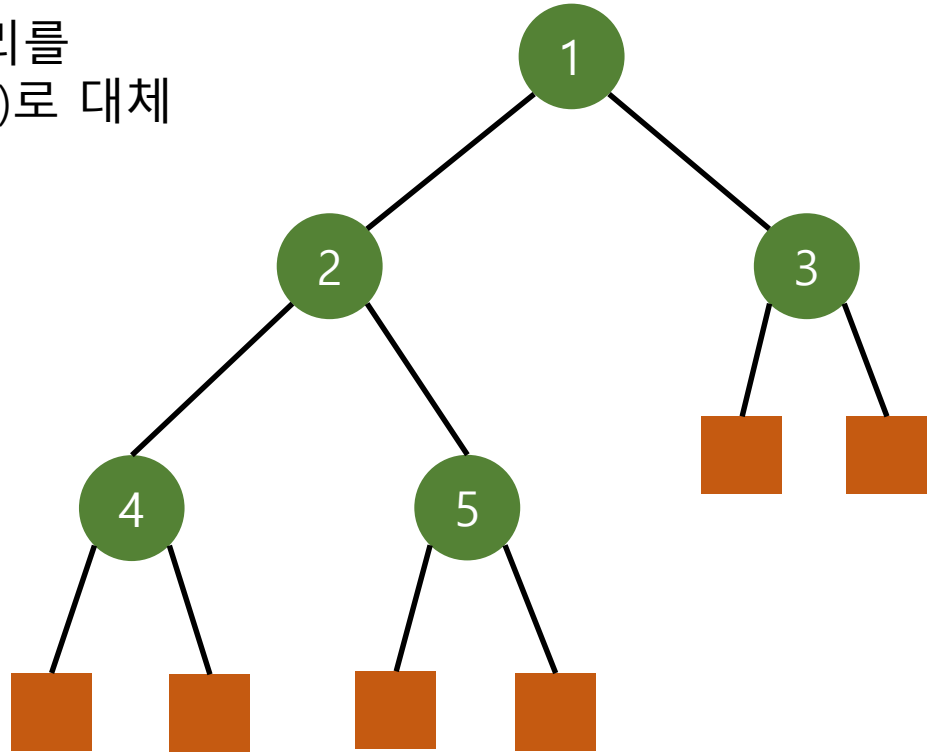


균형 이진 트리(balanced binary tree)의 종류

1. AVL
2. Red black tree
3. B tree
4. B+ tree
5. 2-3 tree
6. 2-3-4 tree

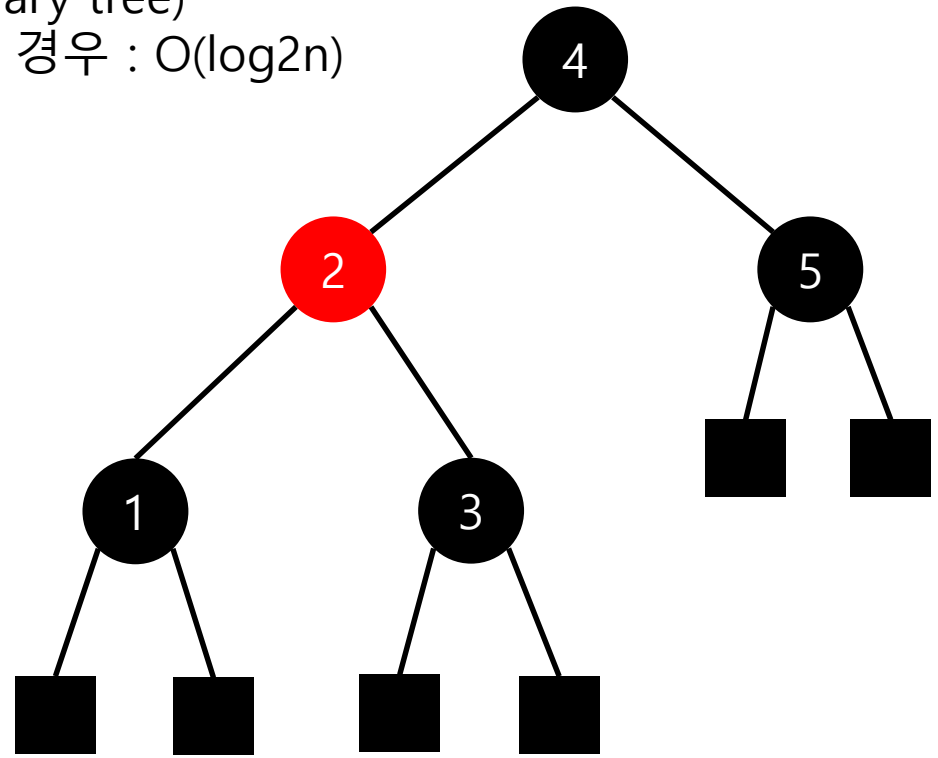
확장 이진 트리(extended binary tree)

모든 공백 이진 서브 트리를
외부 노드(external node)로 대체



레드 블랙 트리(red black tree)

모든 노드의 컬러가 레드 혹은 블랙인 이진 탐색 트리
균형 이진 트리(balanced binary tree)
Insert, search, delete 최악의 경우 : O(log2n)



레드 블랙 트리의 특징

1. 트리의 모든 노드는 레드 아니면 블랙
2. 루트와 외부 노드의 컬러는 블랙
3. 루트에서 외부 노드로의 경로 중에 레드 노드가 연속으로 나올 수 없다.
4. 루트에서 외부 노드로의 모든 경로에서 블랙 노드의 수는 같다.

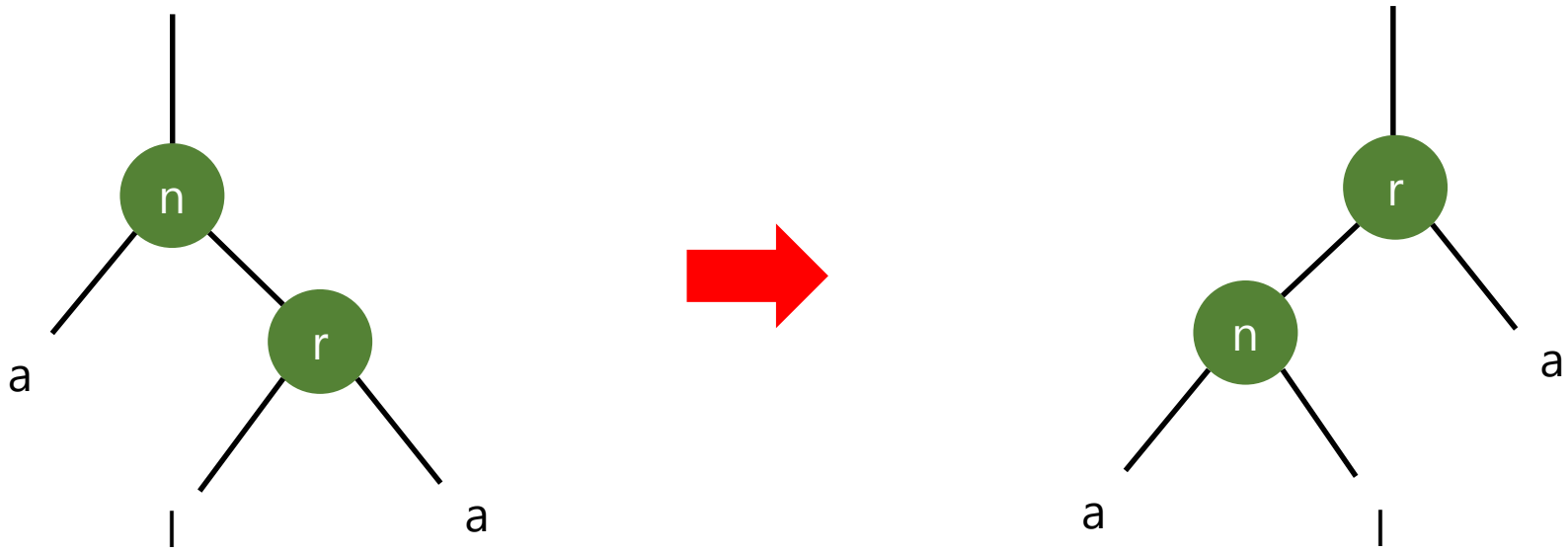
레드 블랙 트리의 높이

N개의 노드가 있는 레드 블랙 트리 높이 <= 2log2(n+1)

RBNode

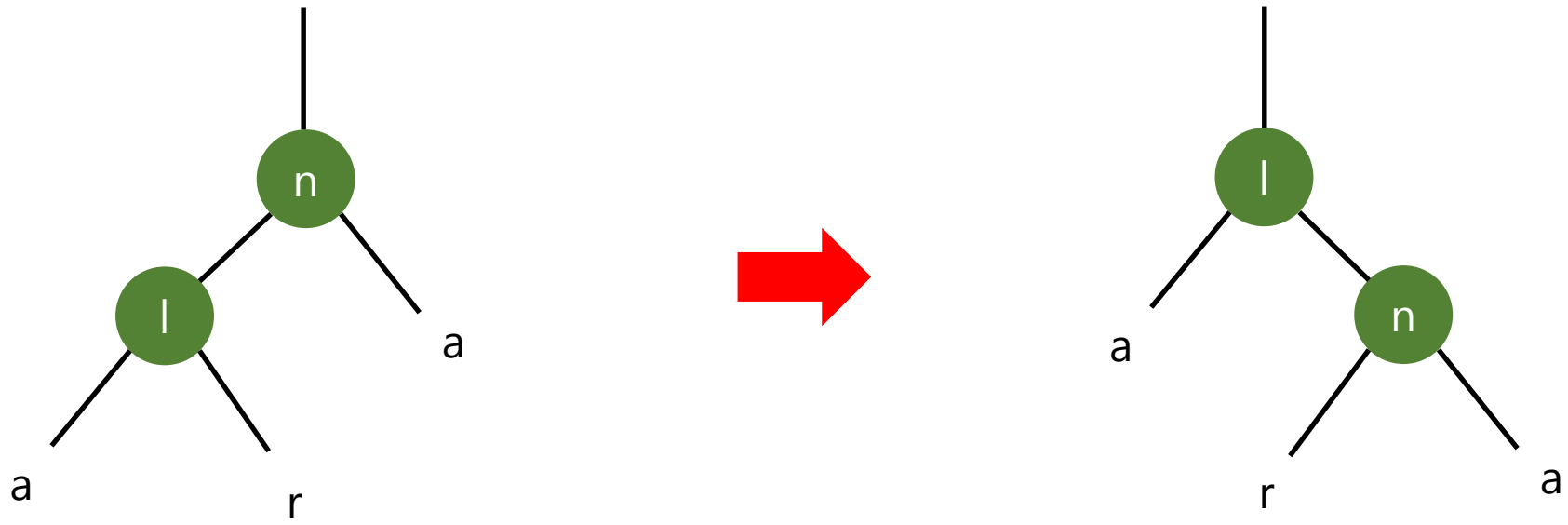
1. key : 트리 내에서 유일한 키
2. color : "RED" 아니면 "BLACK"
3. left_child : 왼쪽 서브 트리
4. right_child : 오른쪽 서브 트리
5. parent : 부모 노드

Left rotation



1. l을 n.right_child로
2. n.parent를 r.parent로
3. n을 r.left_child로

Right rotation



1. r을 n.left_child로
2. n.parent를 l.parent로
3. n을 l.right_child로

insert

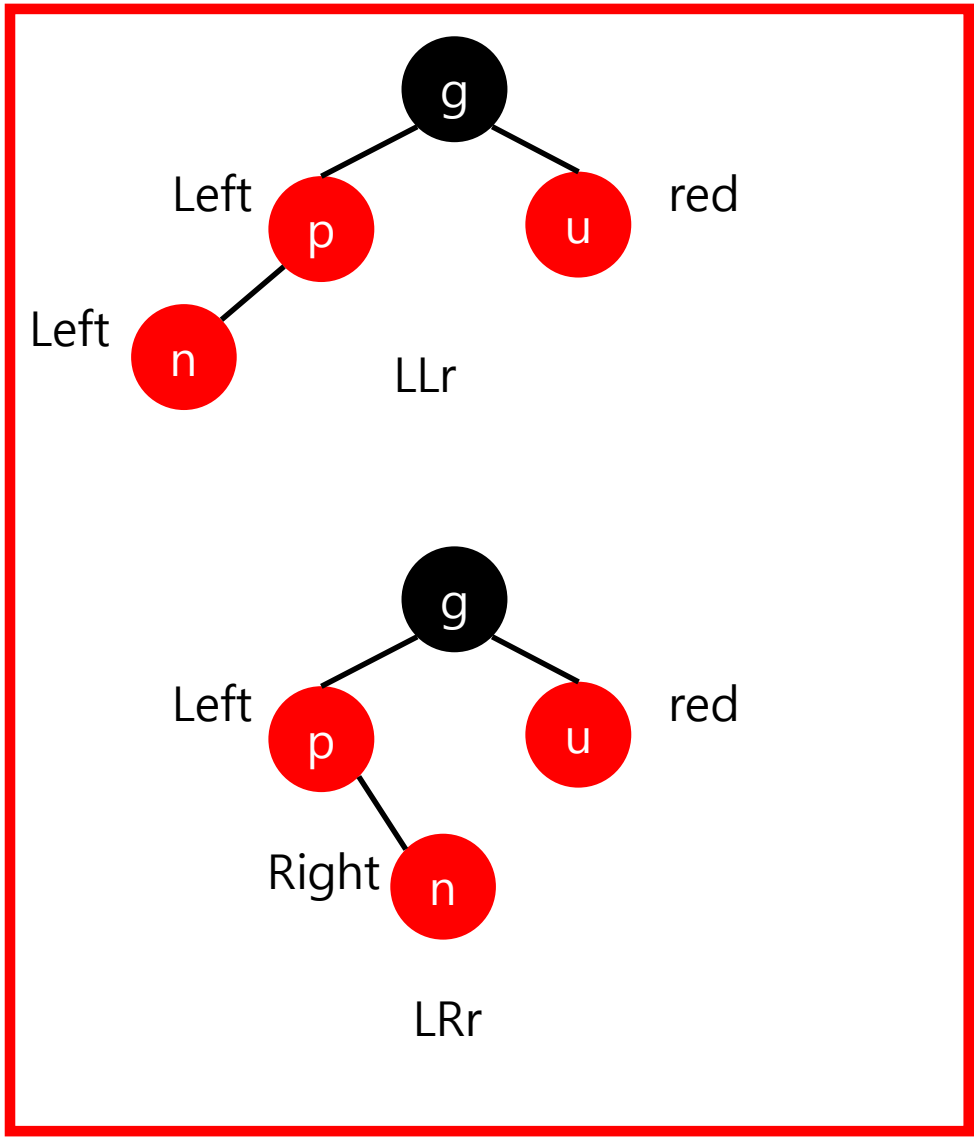
BST에 노드를 삽입한 후
새로운 노드의 색을 RED로 한다
Insert_fix를 호출해 노드를 균형 있게 재배열 한다.

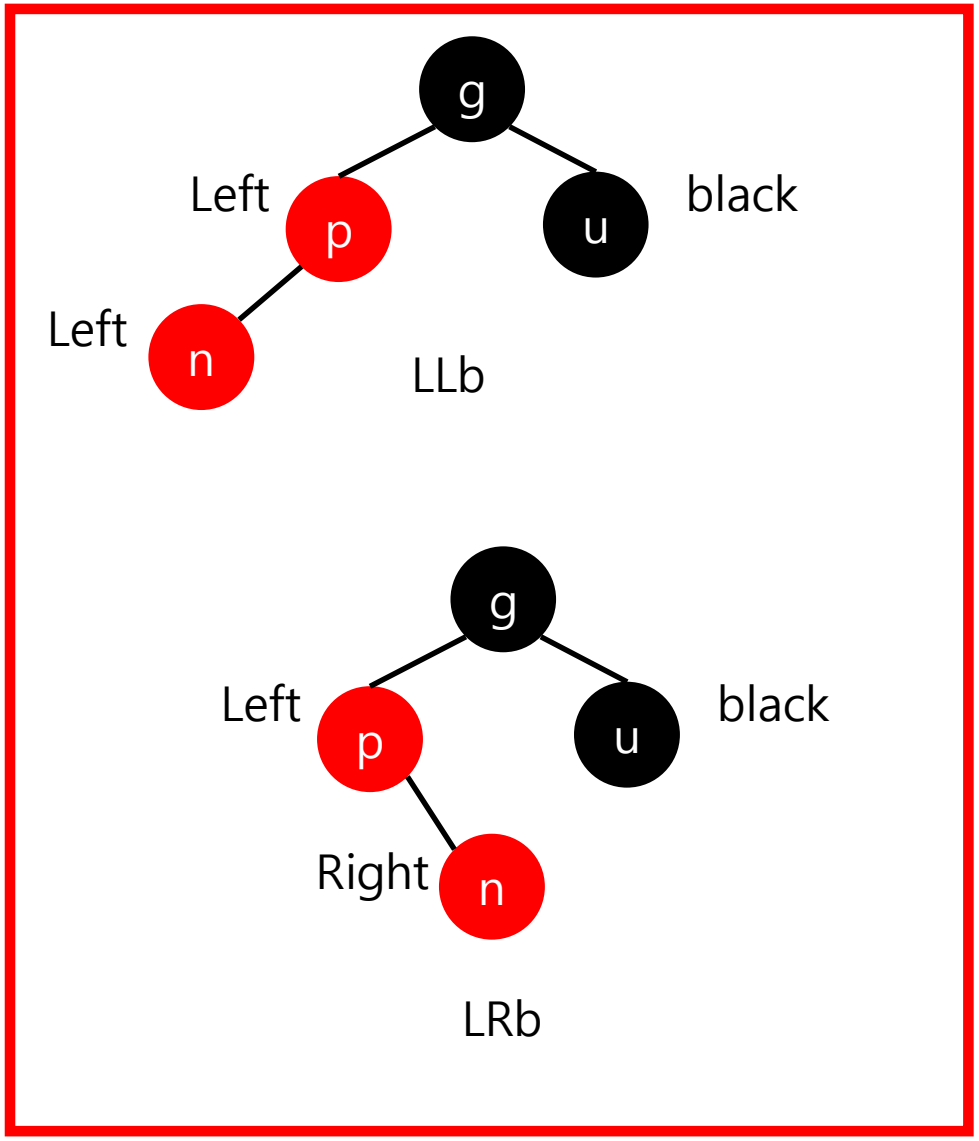
Insert_fix

1. 루트 노드가 RED
   : 루트 노드를 블랙으로 바꾼다.

2. 삽입된 노드와 그 부모 노드가 모두 RED라면
RED 노드가 연속되어 나올 수 없다는 규칙에 위반

불균형 예


g
Left
p
u
red
Left
n
LLr
g
Left
p
u
red
Right
n
LRr


g
Left
p
u
black
Left
n
LLb
g
Left
p
u
black
Right
n
LRb

Insert_fix

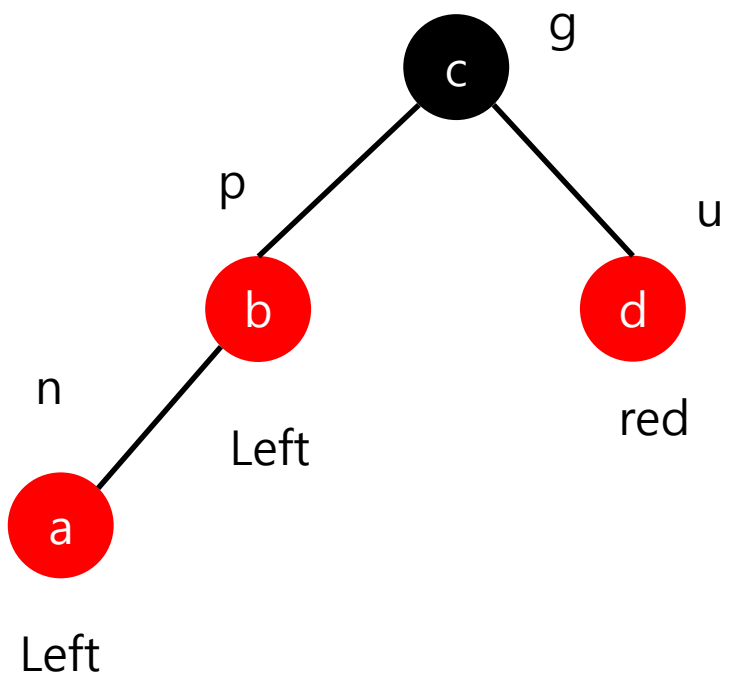
연속된 레드 노드

- 삽입된 노드의 부모 노드가 조부모 노드의 왼쪽 자식일 때
  : LLr, LRr, LLb, LRb

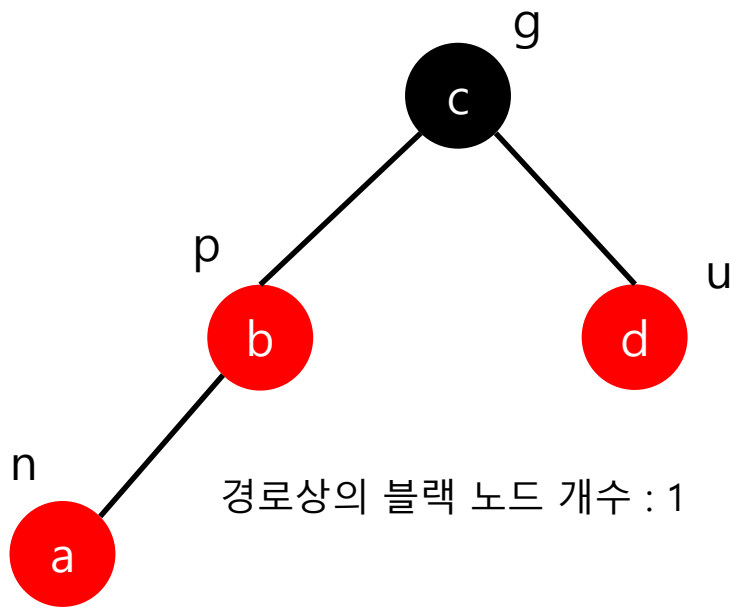
위와 아래의 경우들은 서로 대칭

- 삽입된 노드의 부모 노드가 조부모 노드의 오른쪽 자식일 때
  : RLr, RRr, RLb, RRb

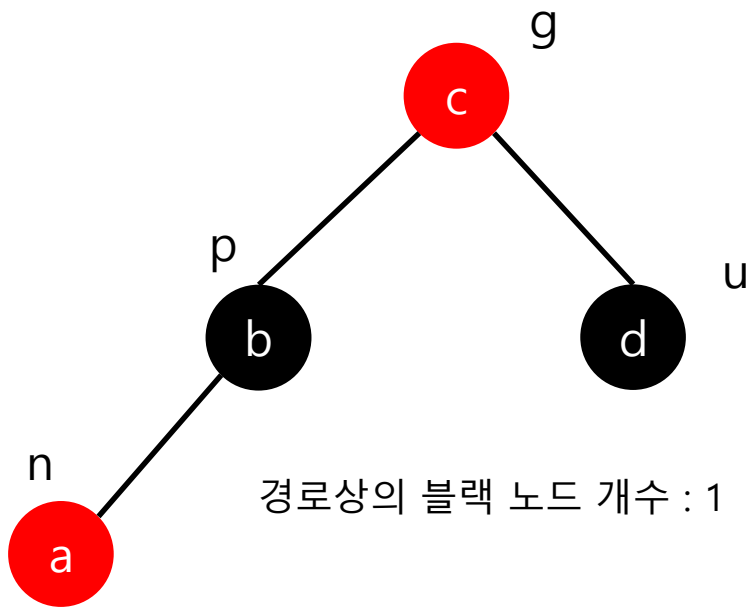
XYr
: 부모의 형제가 RED



XYr
: 부모의 형제가 RED



경로상의 블랙 노드 개수 : 1



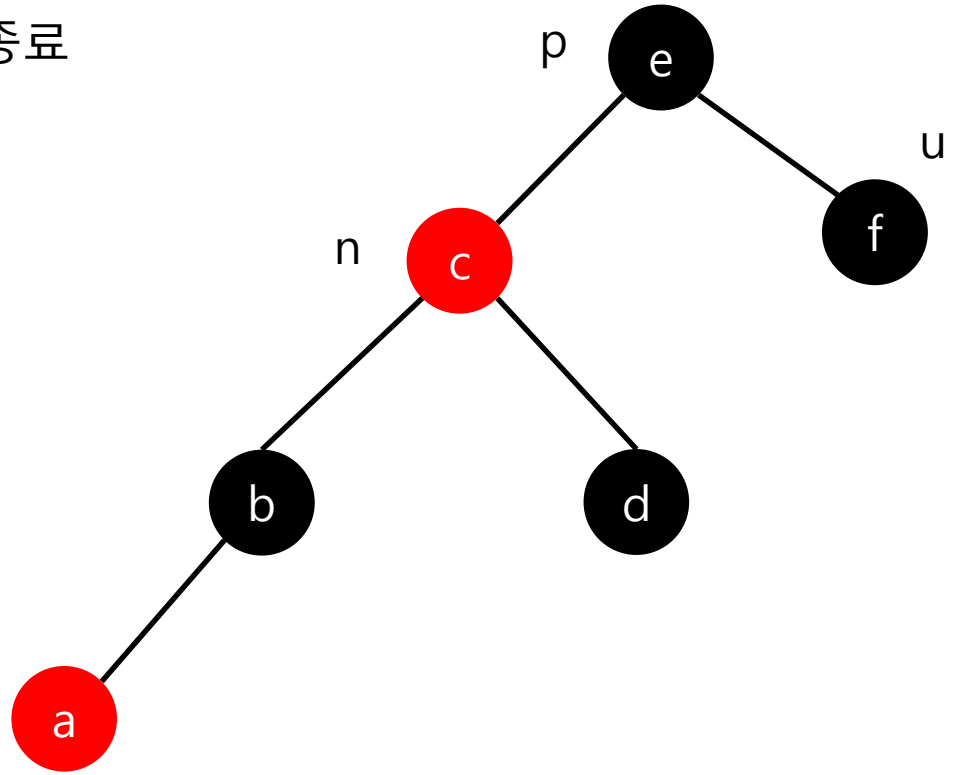
경로상의 블랙 노드 개수 : 1

컬러의 변경!!

1. 조부모를 RED로
2. 부모와 부모 형제를 BLACK으로

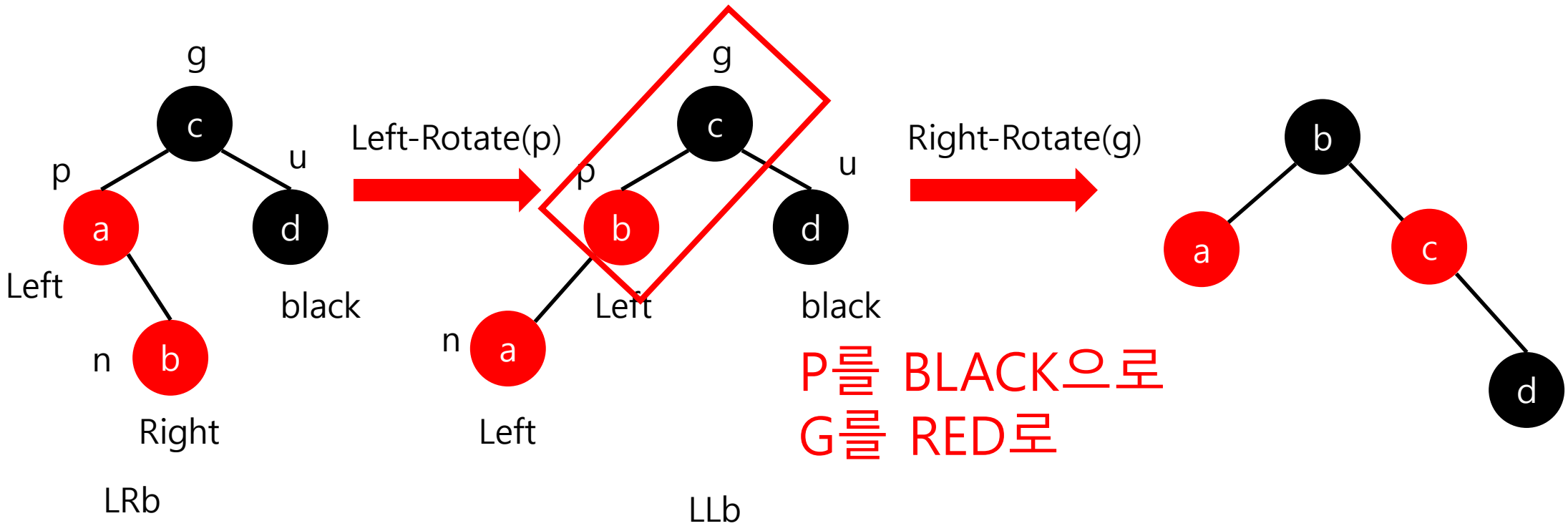
XYr
: 부모의 형제가 RED

새로운 p가 BLACK이면 종료
만약 RED이면 반복 실행

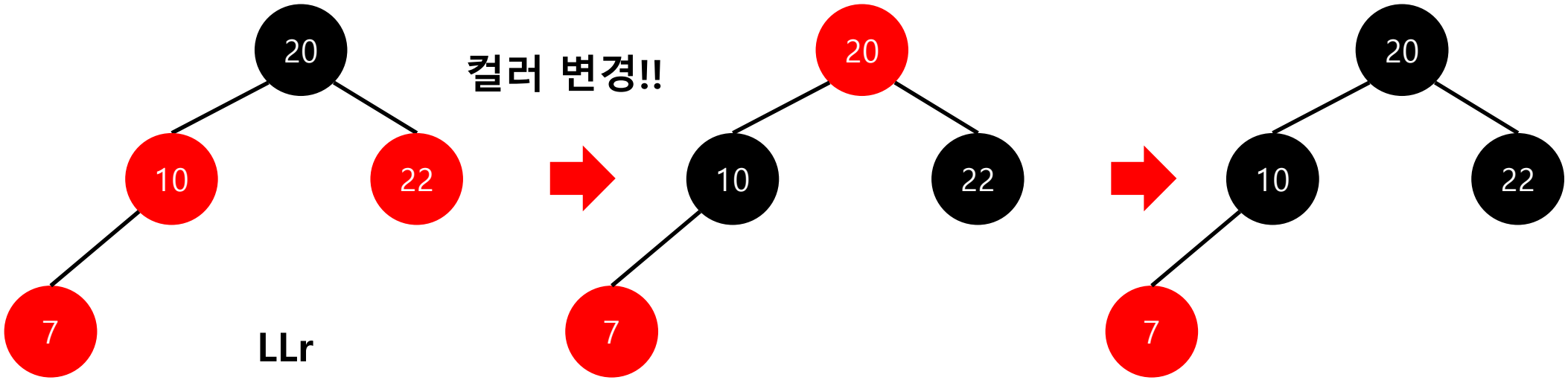


XYb

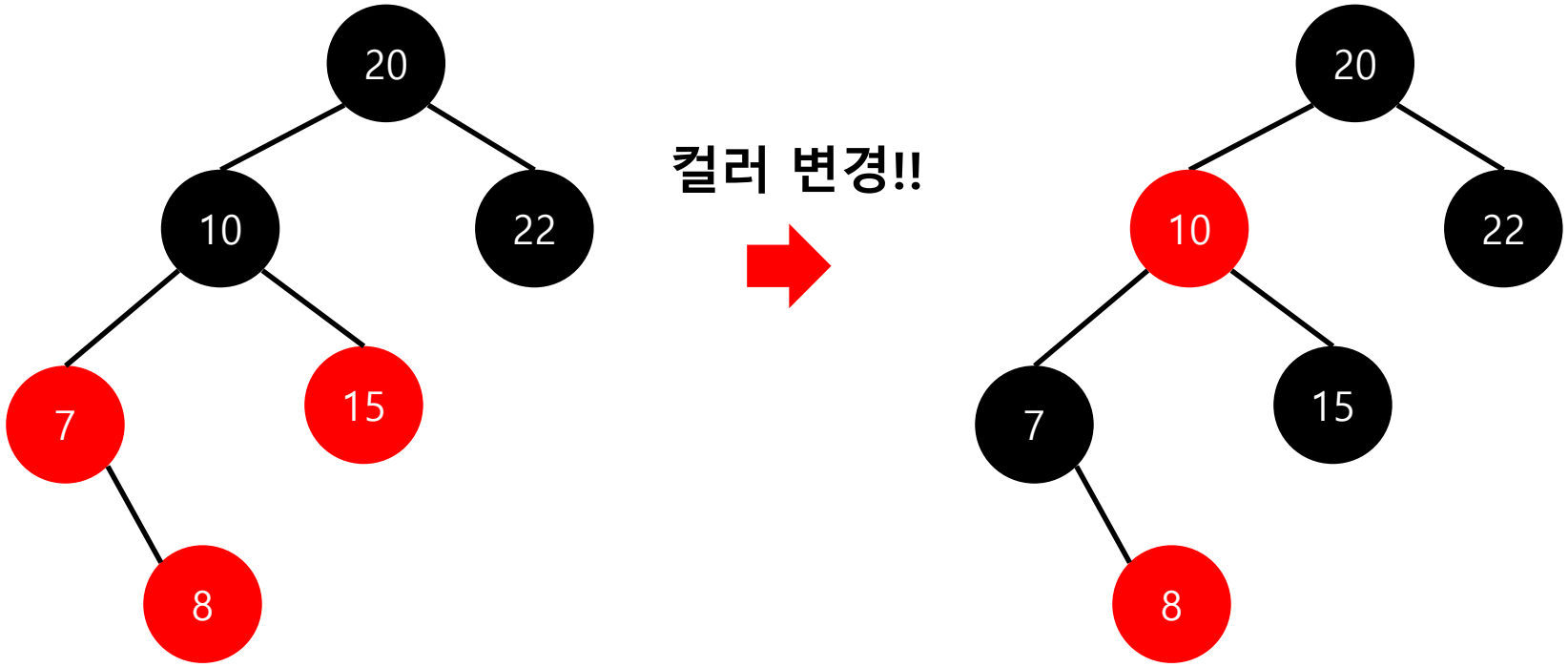
: 부모의 형제가 BLACK



Insert 예제

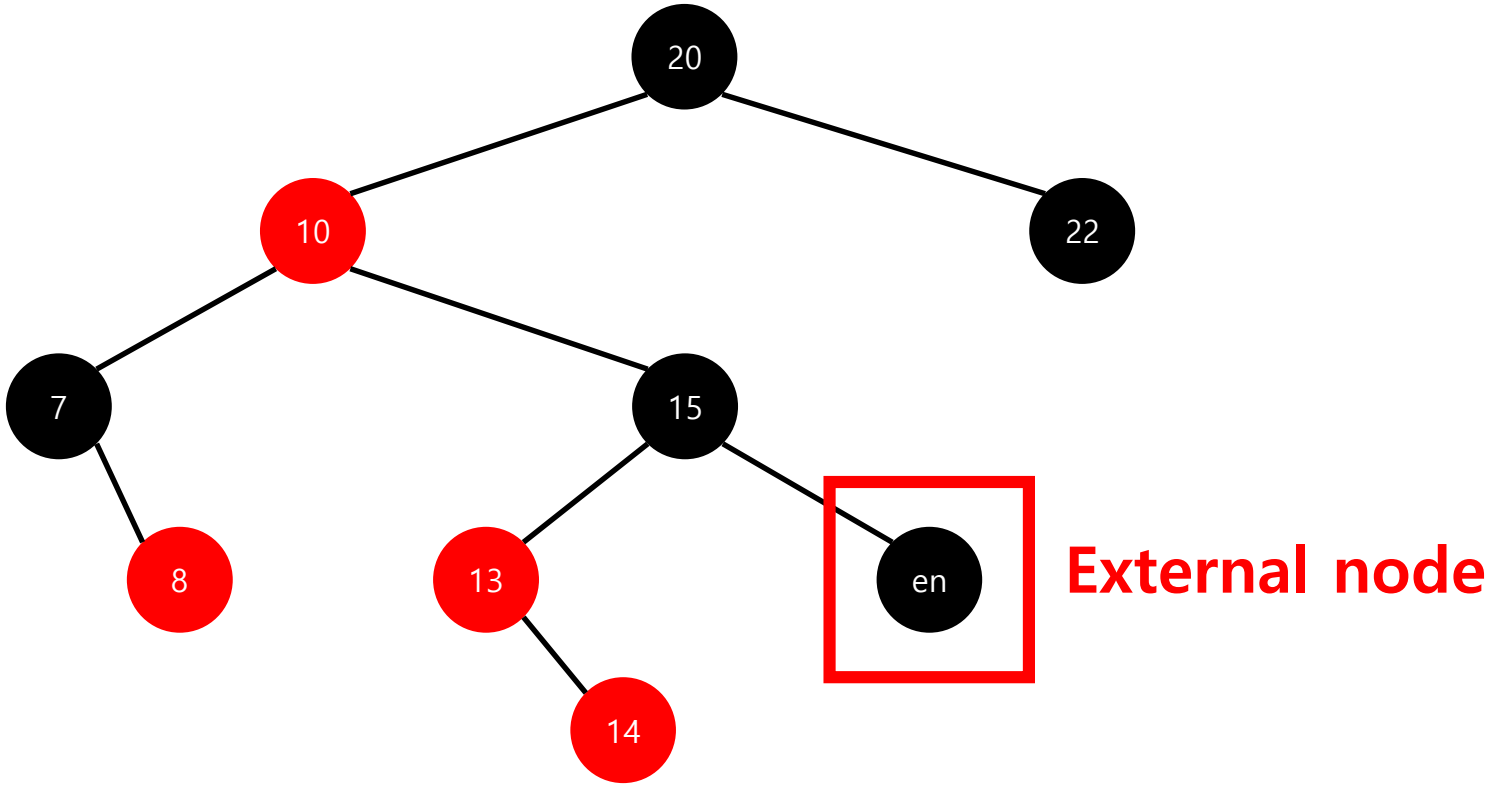


Insert 예제



LRr

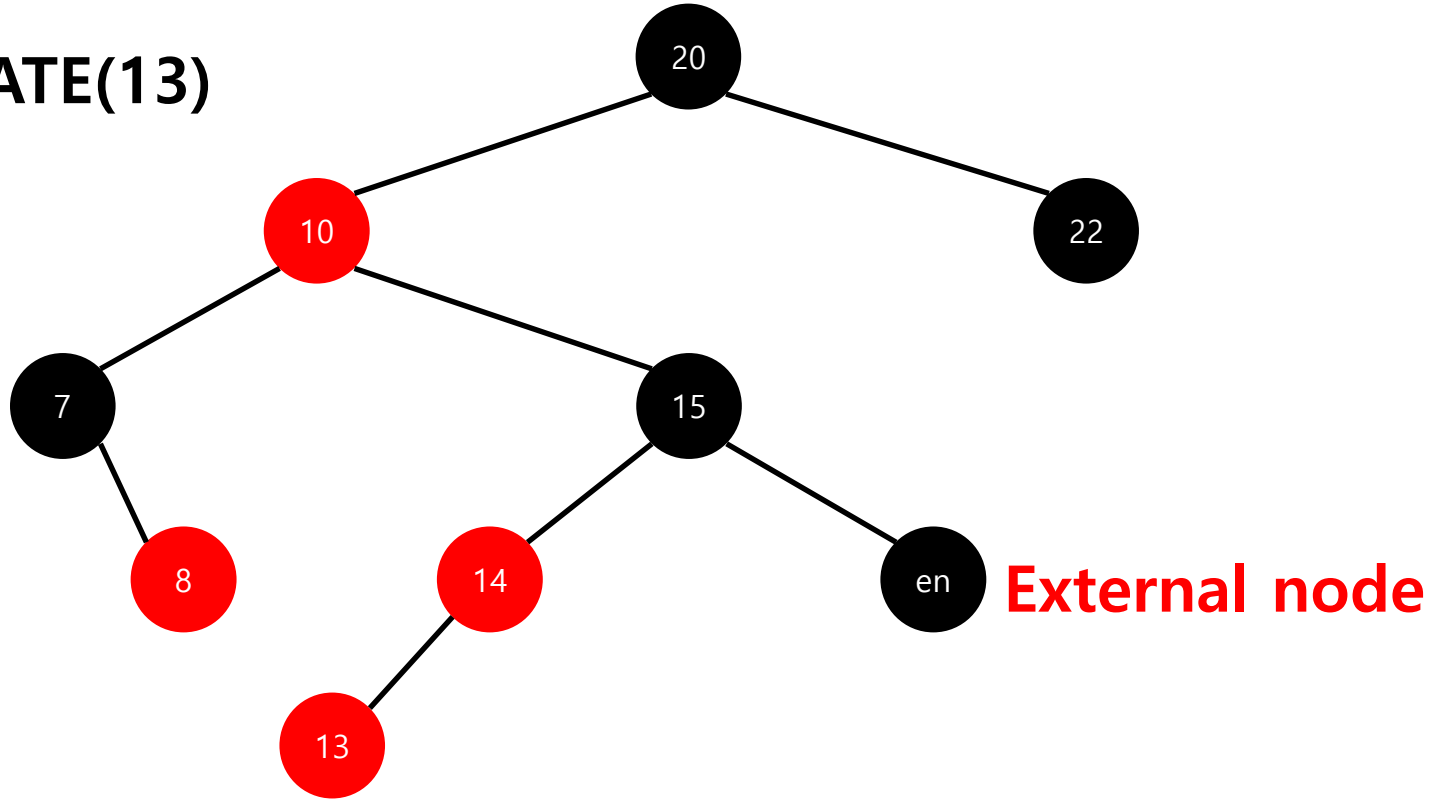
Insert 예제



LRb

Insert 예제

LEFT-ROTATE(13)

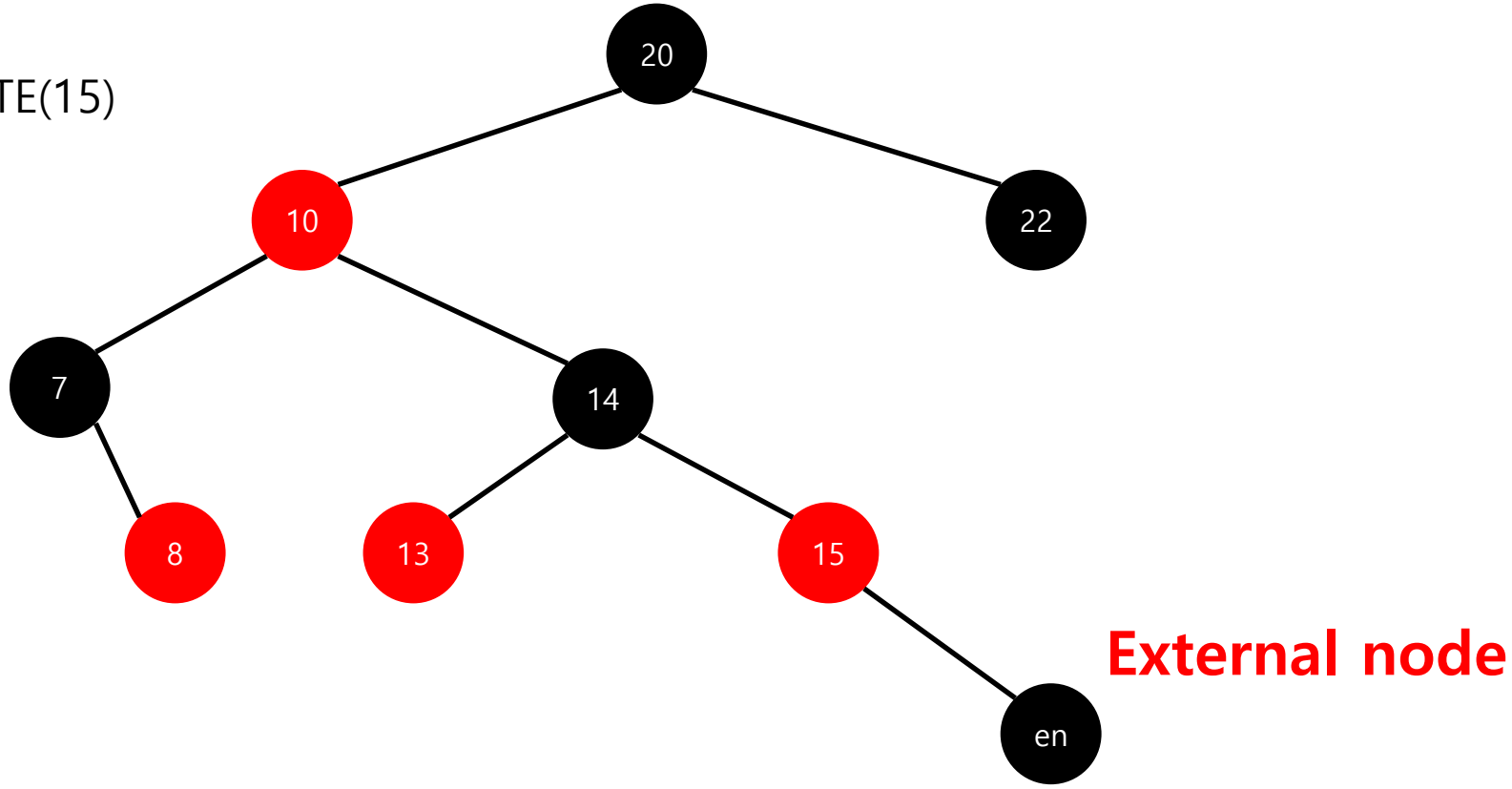


External node

LRb → LLb

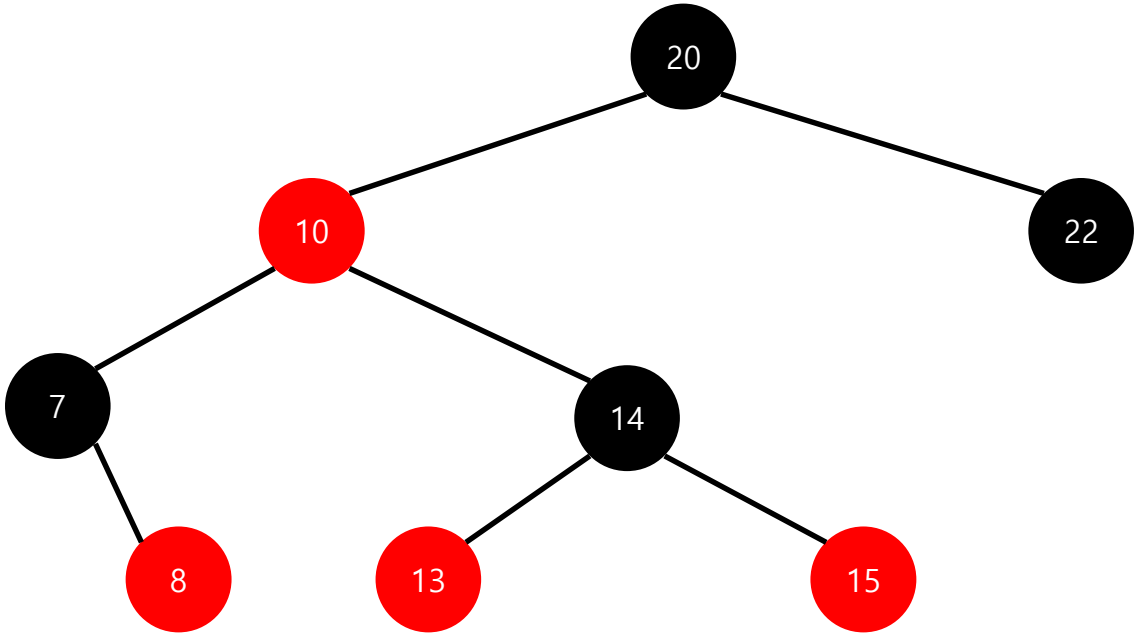
Insert 예제

색 변경 후
RIGHT-ROTATE(15)



External node

Insert 예제



delete

BST에서 노드를 삭제한 후
삭제된 노드가 BLACK이라면
delete_fix를 호출해 노드를 균형 있게 재배열 한다.

delete_fix

1. 삭제된 노드가 루트, 새로운 루트가 RED
   : 루트 노드를 블랙으로 바꾼다.
2. 삭제된 노드가 BLACK이므로 루트에서 외부 노드까지 모든 경로의 블랙 노드의 수는 같다는 규칙에 위반

delete_fix

경로 상의 블랙 노드 수가 하나 적을 때
1) 삭제된 노드의 자식 노드가 RED
: BLACK으로 바꾼다(EASY)
2) 삭제된 노드의 자식 노드가 BLACK
- 자식 노드에 extra BLACK을 준다(DIFFICULT)
- 구현 주의 사항
: 삭제된 노드의 자식 노드가 외부 노드일 수 있음!!

delete_fix

Extra Black을 루트 쪽으로
- Extra black을 가진 노드가 RED이면
  이 노드를 BLACK으로 변경
- 루트에 도달하면 extra BLACK을 없앤다

delete_fix

삭제된 노드의 자식 노드 c가 왼쪽 자식 노드일 때

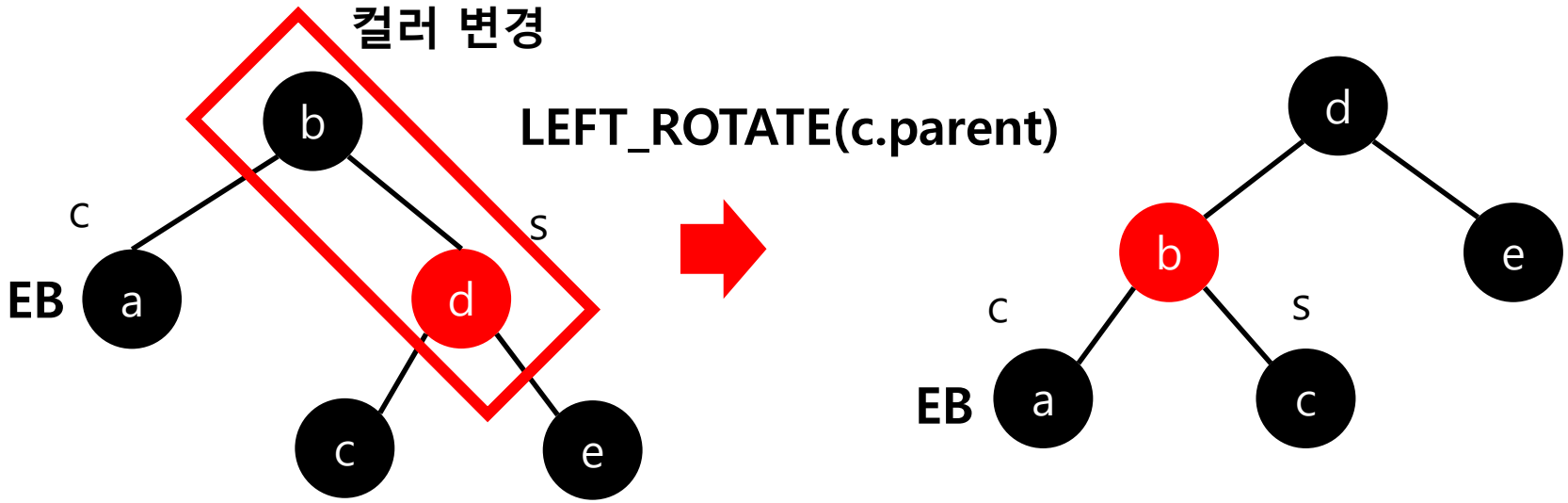
위와 아래의 경우는 대칭

삭제된 노드의 자식 노드 c가 오른쪽 자식 노드일 때

delete_fix

삭제된 노드의 자식 노드 c가 왼쪽 자식일 때
- 노드 c의 형제 노드 s가 RED일 때
- 노드 c의 형제 노드 s가 BLACK일 때
    1) 노드 s의 두 자식이 BLACK
    2) 노드 s의 왼쪽 자식이 RED
    3) 노드 s의 오른쪽 자식이 RED

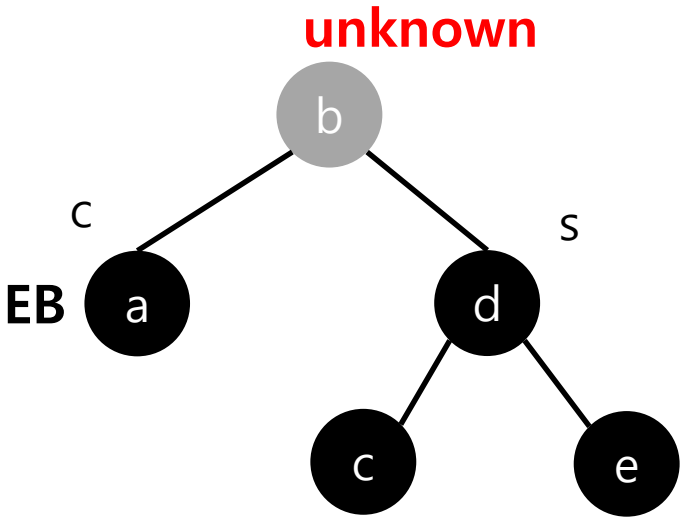
S가 RED

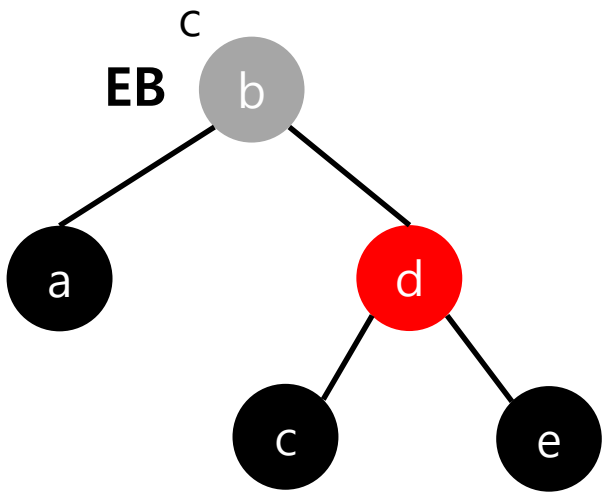


S가 BLACK인 경우가 됨

1. c.parent <- RED, s <- BLACK
2. LEFT-ROTATE(c.parent)

S가 BLACK
: s의 두 자식 모두 BLACK

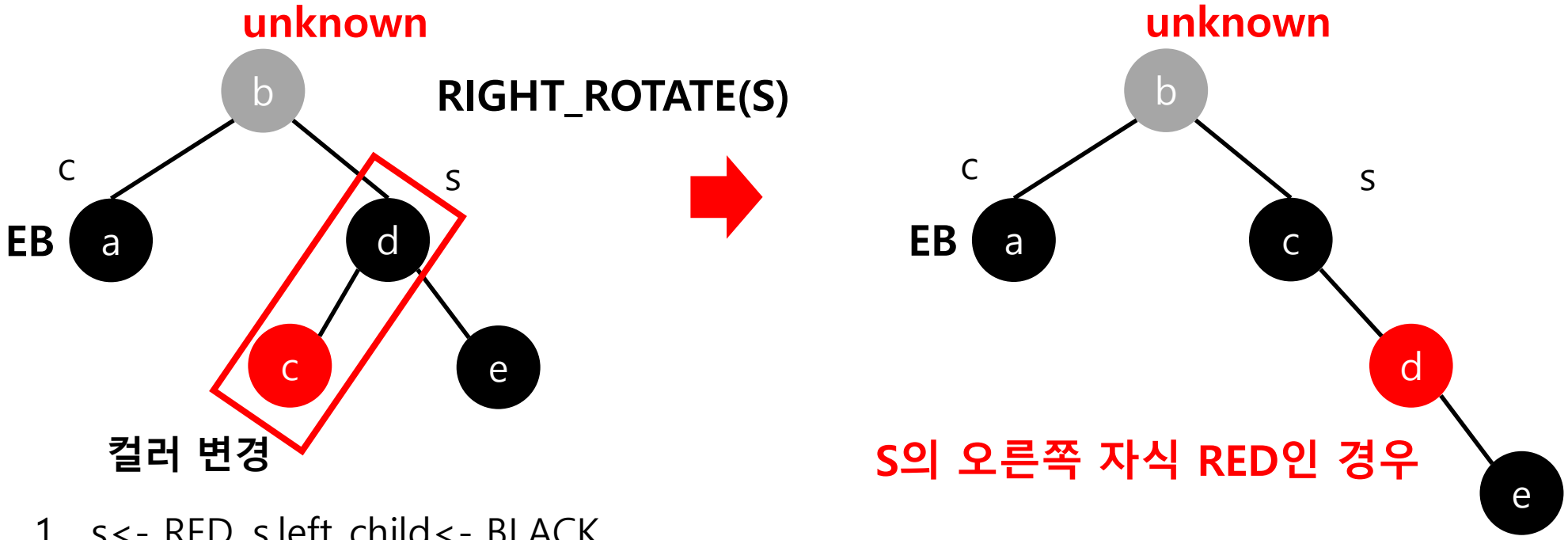




1. 노드 c와 노드 s에서 BLACK 하나씩 추출
2. C.parent에게 extra BLACK 줌
3. C.parent가 새로운 C

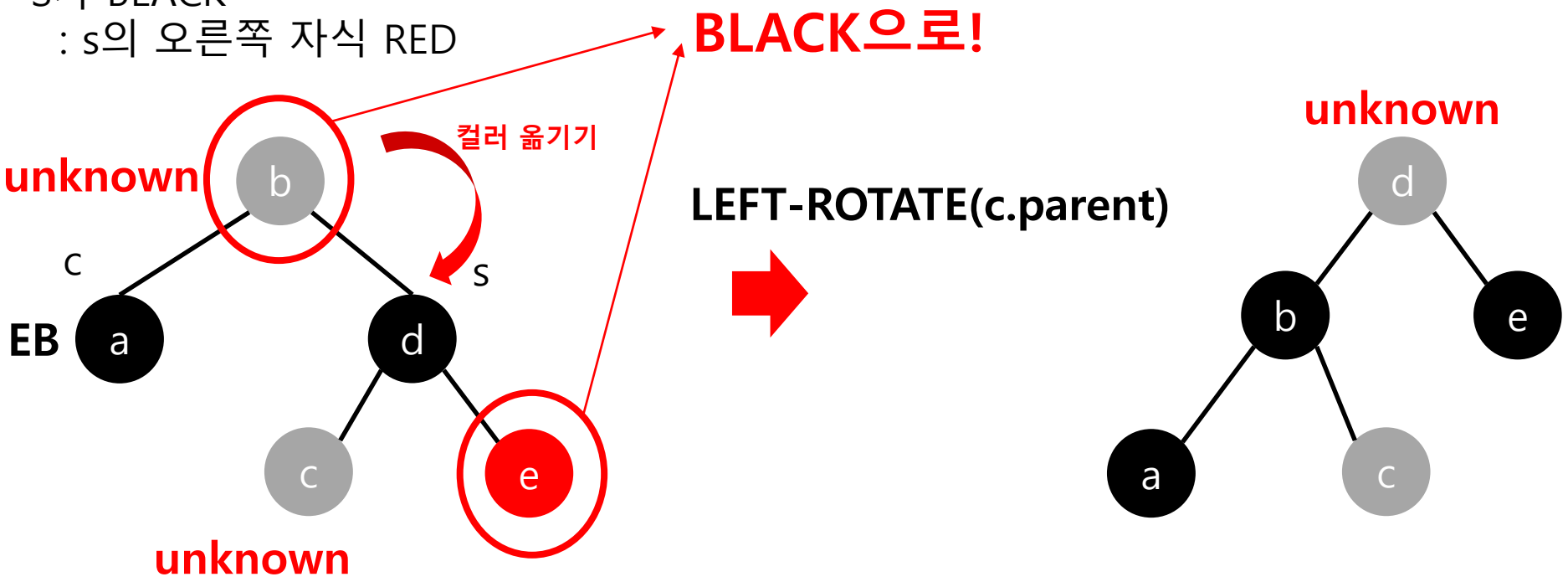
새로운 노드 C가 BLACK이면 다시 더블 블랙
C가 RED이면 BLACK으로 만들고 종료

S가 BLACK
: s의 왼쪽 자식 RED



1. s<- RED, s.left_child<- BLACK
2. RIGHT_ROTATE(S)

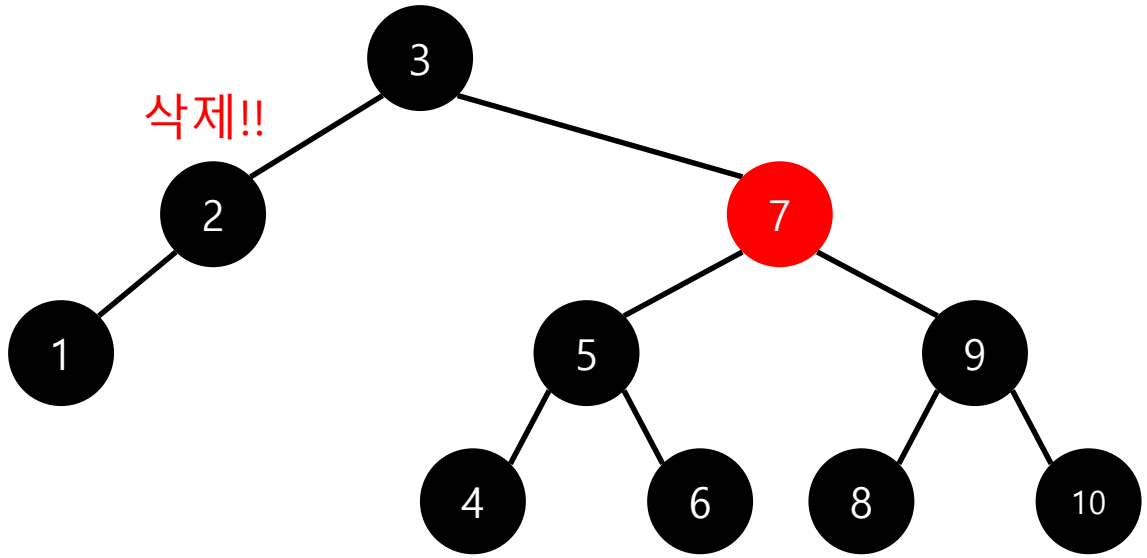
S가 BLACK
: s의 오른쪽 자식 RED



1. S의 색을 c.parent 컬러로
2. C.parent<-BLACK, s.right<-BLACK
3. LEFT-ROATE(c.parent)
4. Extra BLACK 없애고 종료

Delete 예제

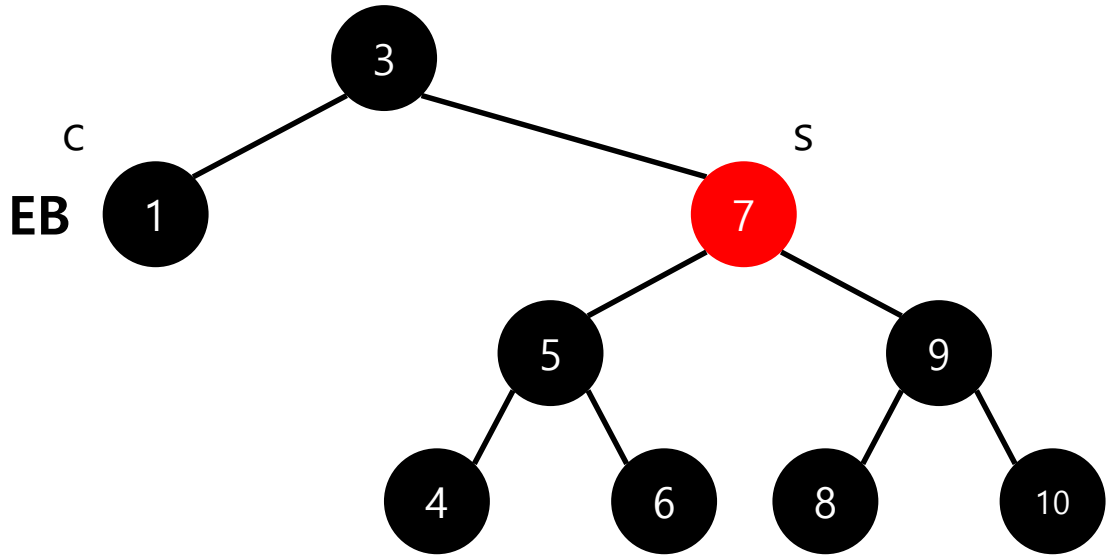
case1, case 2-1



Delete 예제

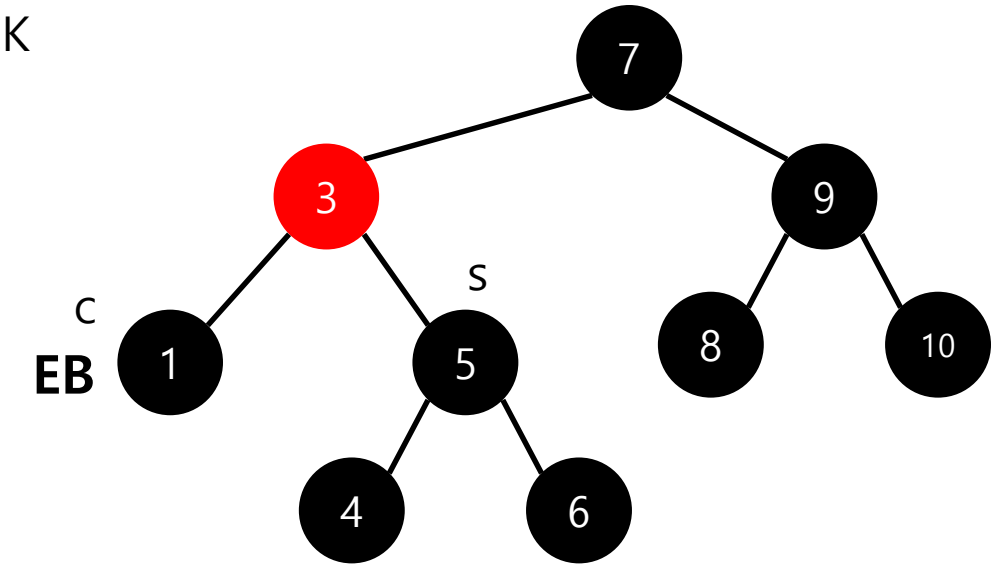
case1, case 2-1

CASE 1 : S가 RED



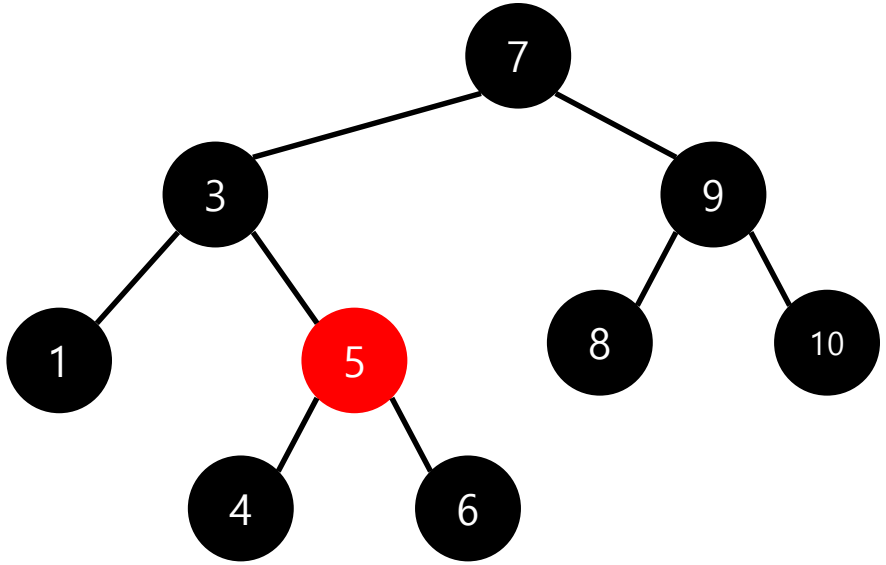
Delete 예제
case1, case 2-1

CASE 2-1 : S가 BLACK,
S.LEFT, S.RIGHT -> BLACK



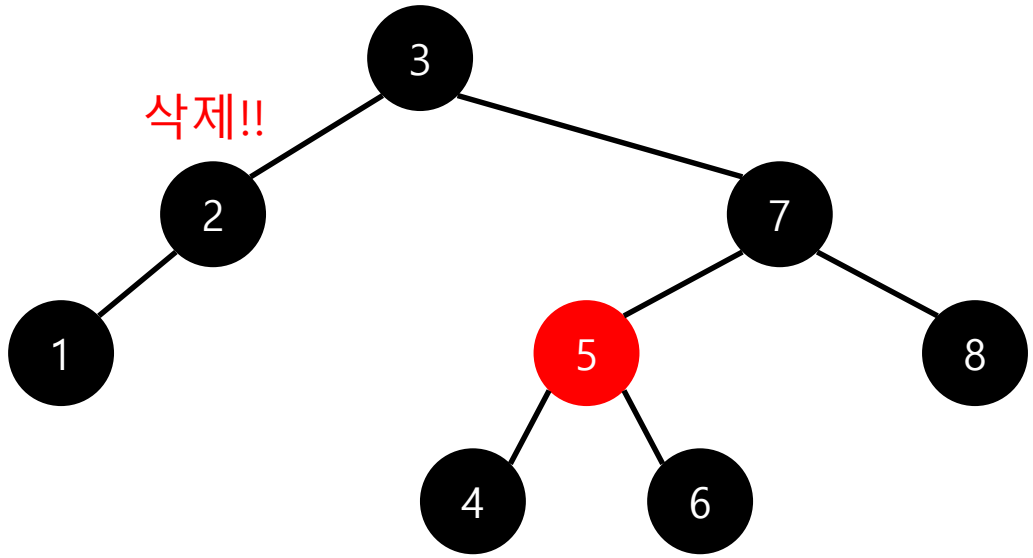
Delete 예제

case1, case 2-1



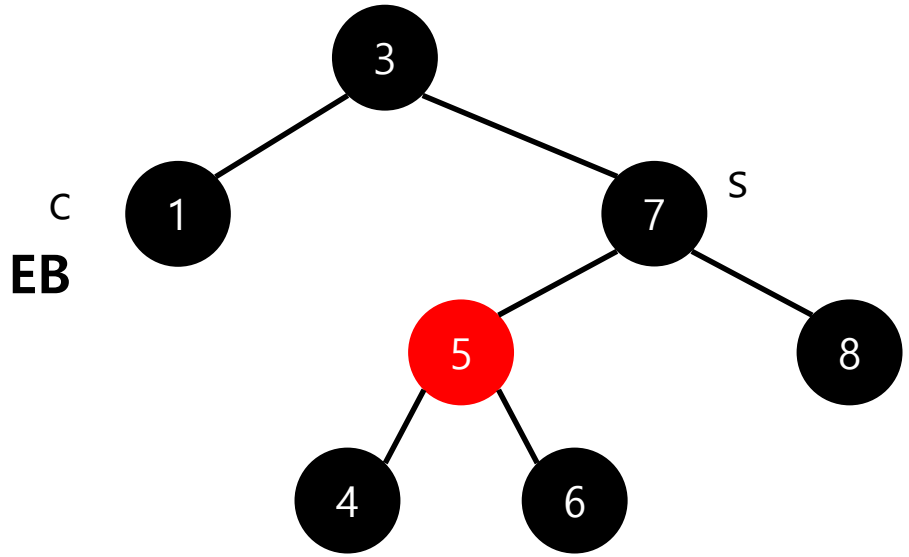
Delete 예제

case2-2, case 2-3



Delete 예제
case2-2, case 2-3

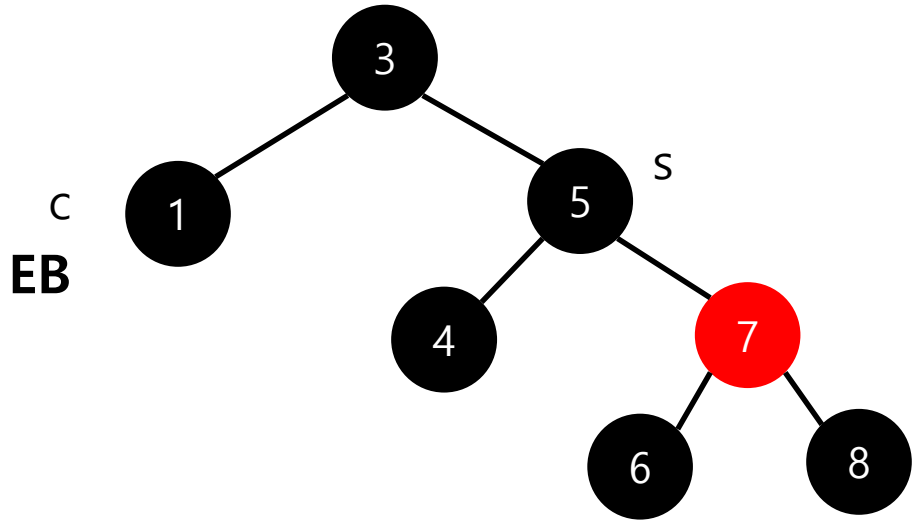
CASE 2-2 : S가 BLACK,
S.LEFT -> BLACK



Delete 예제

case2-2, case 2-3

CASE 2-3 : S가 BLACK,
S.RIGHT -> BLACK



Delete 예제

case2-2, case 2-3