09장 Red-Black Tree

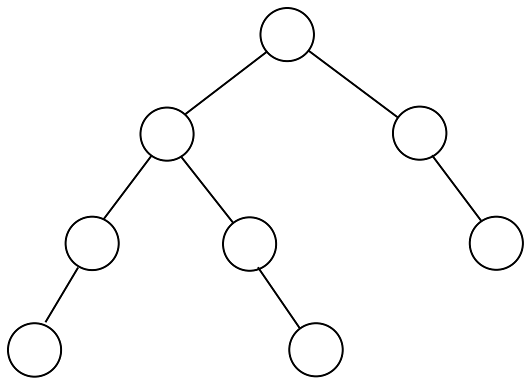
레드블랙 트리

- **자가균형이진탐색 트리 self-balancing binary search tree**

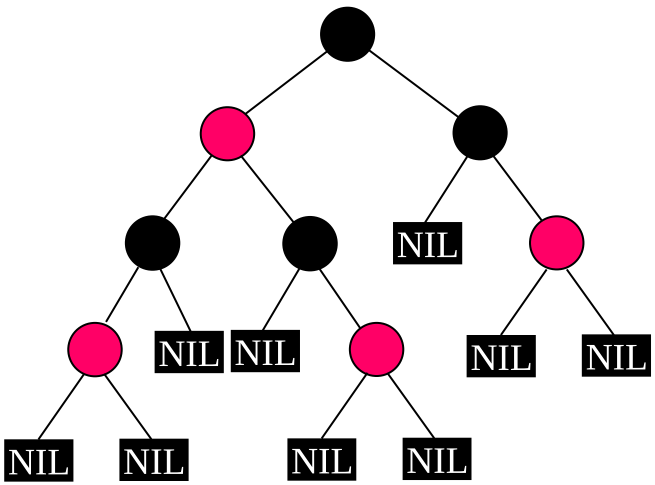
- **이진검색트리의 각 노드는 red 또는 black의 색을 갖는다**
- **다음의 레드블랙특성을 만족해야 한다**
  ① Root Property : 루트는 black 이다
  ② External Property : 모든 리프(NIL)는 black 이다
  ③ Internal Property : 노드가 red 이면 그 노드의 자식들(2개 노드)은 반드시 black 이다
  → No Double Red(빨간색 노드가 연속으로 나올 수 없다.)
  ④ Depth Property : 루트 노드에서 임의의 리프 노드에 이르는 경로에서 만나는 black 노드의 수는 모두 같다
  → 이를 **black-height** 라고 한다

- 리프 노드: 이진검색트리의 노드가 가진 두 개의 자식 포인터 중 NIL인 것.
  - ✓ 일반적인 의미의 리프 노드와 다르다.
  - ✓ 모든 NIL 포인터가 NIL이라는 리프 노드를 가리킨다고 가정한다
- 내부 노드: NIL 노드가 아닌 일반 노드

이진검색트리를 레드블랙트리로 만든 예



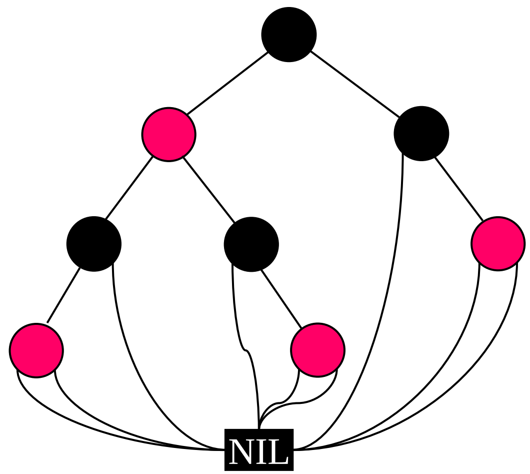
(a) 이진검색트리의 한 예



(b) (a)를 레드블랙트리로 만든 예

- 루트 노드는 블랙으로 칠이 되어 특성 1번을 만족한다.
- 임의의 노드에서 자식이 없는 쪽은 모두 NIL 리프를 붙이고, 모든 NIL 노드는 블랙으로 칠하여 특성 2번을 만족한다.
- 임의의 노드가 레드이면 그 자식은 반드시 블랙이라는 특성 3번을 만족한다.
- 루트에서 모든 리프 노드에 이르는 경로상에서 만나는 블랙 노드의 수는 항상 2개로 똑같아 특성 4번을 만족한다.

이진검색트리를 레드블랙트리로 만든 예



(c) 실제 구현시의 NIL 노드 처리 방법

- 노드 하나를 할당하여 이를 리프로 정하고 모든 NIL 리프에 대한 포인터가 이 노드를 가리키도록 한다.
- 공간도 절약 및 경계 조건을 다루기가 편리해진다.

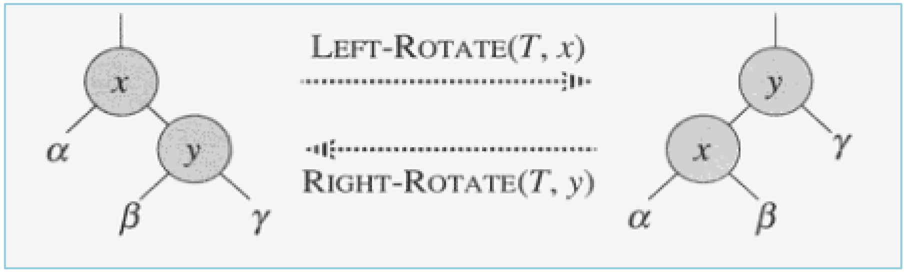
레드블랙 트리

- 레드블랙트리의 특성을 만족하면 갖게 되는 가장 중요한 특성
  - 루트 노드부터 가장 먼 경로까지의 거리가,
    가장 가까운 경로까지의 거리의 두 배보다 항상 작다.
  - 즉, 완전하지는 않지만 균형이 잡혀 있다.

  - 그러므로, 삽입, 삭제, 검색 시
    최악의 경우(worst-case)에서의 시간복잡도가
    트리의 높이에 따라 결정되기 때문에
    보통의 이진검색 트리에 비해 효율적이다.

레드블랙 트리 왼쪽 회전 / 오른쪽 회전

- **Left Rotation / Right Rotation**
  - 이진탐색트리의 특성을 유지하면서 회전한다.
- **x를 중심으로 왼쪽 회전**
  - y가 x의 부모가 된다.
  - y의 왼쪽 자식은 x의 오른쪽 자식이 된다.
- **y를 중심으로 오른쪽 회전**
  - x가 y의 부모가 된다.
  - x의 오른쪽 자식은 y의 왼쪽 자식이 된다.



레드블랙 트리의 높이

- h(x): 노드 x의 높이
  - 자신으로부터 리프 노드까지 가장 긴 경로에 포함된 edge 개수
- bh(x): black-height
  - x로부터 리프 노드까지 경로상의 블랙노드 개수 (노드x자신이 black이면, 포함하지 않음)
  - 조건 ④에 의해 레드노드가 연속될 수 없으므로
    - ✓ $bh \geq h / 2$
- root x의 서브트리의 내부노드: $(2^{bn(x)} - 1)$개
- n 개의 내부노드를 갖는 레드블랙 트리의 높이:
  - $2\log n + 1$, $n \geq 2^{bh(x)} - 1 \geq 2^{h/2} - 1$ 이므로,
  - $O(\log n)$

레드블랙트리에서의 삽입

1. BST(이진검색트리)의 삽입알고리즘으로 새노드 x를 삽입한다
2. 새노드 x의 색을 우선 red로 칠한다
3. 3가지 경우로 나누어 처리한다

- 새 노드는 항상 맨 아래쪽에 매달리므로 삽입 직후에, x의 아래쪽은 블랙 노드인 리프 두개만이 있으으므로 레드블랙특성을 위반하지 않는다.
- 그러므로, x의 위쪽에 문제가 발생하는지 확인하여야 한다.
- p(x의 부모)가 블랙이면 완료

레드블랙트리에서의 삽입

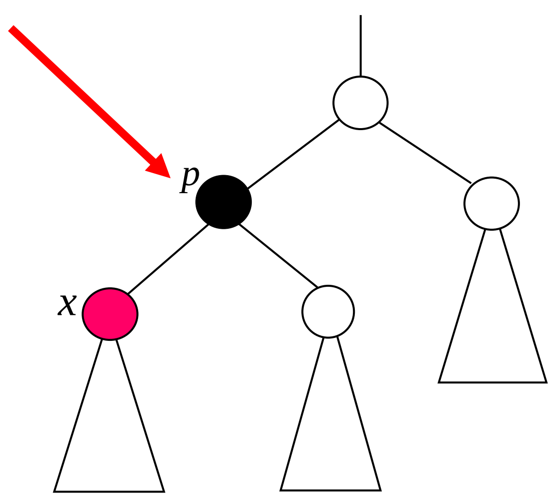
- p가 레드이면, 특성 ③위반.
- p^2(p의 부모)는 반드시 black이다
  - 이유: 새노드 삽입이전에 레드블랙특성을 만족하였다.
  - 그러므로, x의 형제노드도 반드시 black이다.
  - x 주변에 레드나 블랙 두가지 모두 가능한 것의 p의 형제노드(x의 삼촌, 이를 s라 하자)이다.
  - s의 색상에 따라서
    - Case 1: s가 red
    - Case 2: s가 black
      - Case 2-1: x가 p의 오른쪽 자식
      - Case 2-2: x가 p의 왼쪽 자식

레드블랙트리에서의 삽입

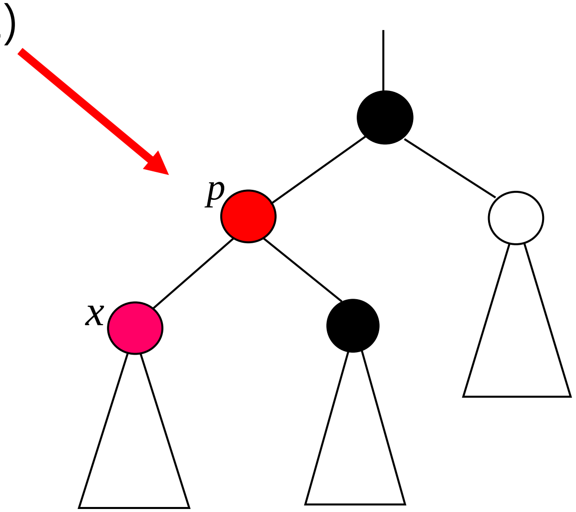
- 만일 *x*의 부모 노드 *p*의 색상이
  1) 블랙이면 아무 문제 없다.
  2) 레드이면 레드블랙특성 ③이 깨진다.

그러므로
*p*가 레드인 경우만
고려하면 된다

1)

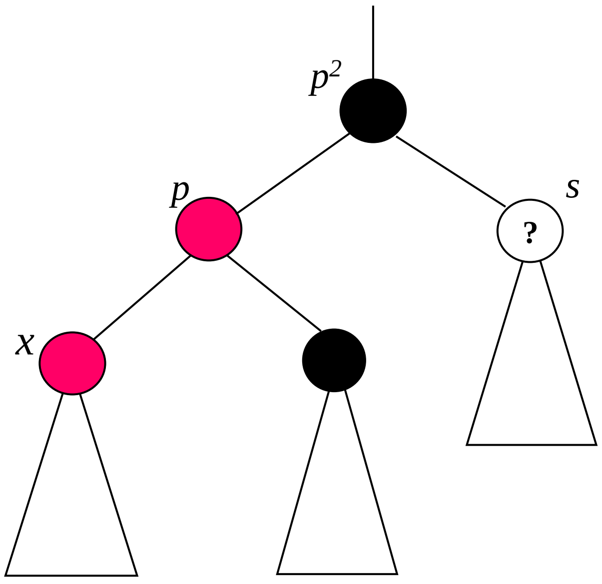


2)



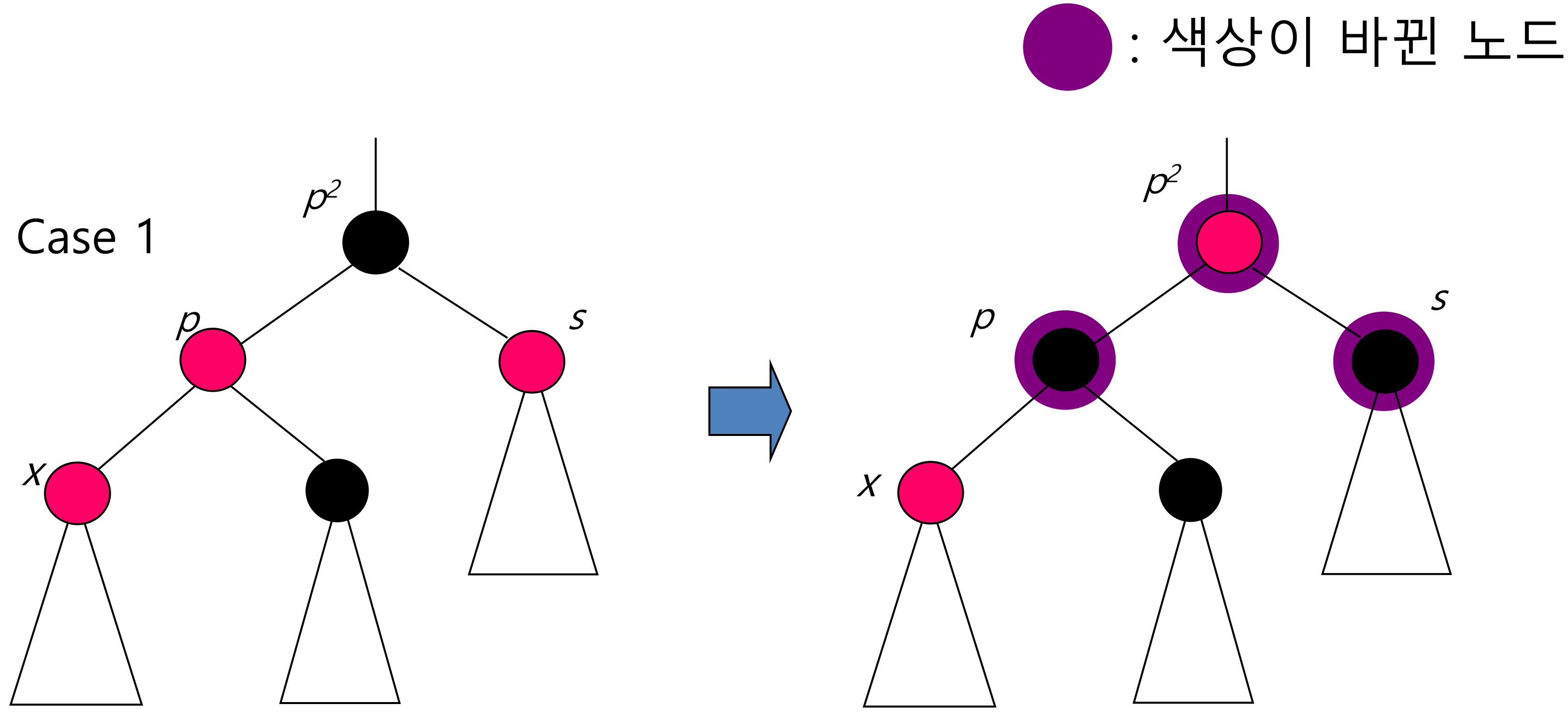
레드블랙트리에서의 삽입

- p^2와 x의 형제 노드는 반드시 블랙이다
- s의 색상에 따라 두 가지로 나눈다
  - Case 1: s가 레드
  - Case 2: s가 블랙



레드블랙트리에서의 삽입

- **Case 1: *s*가 레드**



p^2에서 방금과 같은 문제가 발생할 수 있다

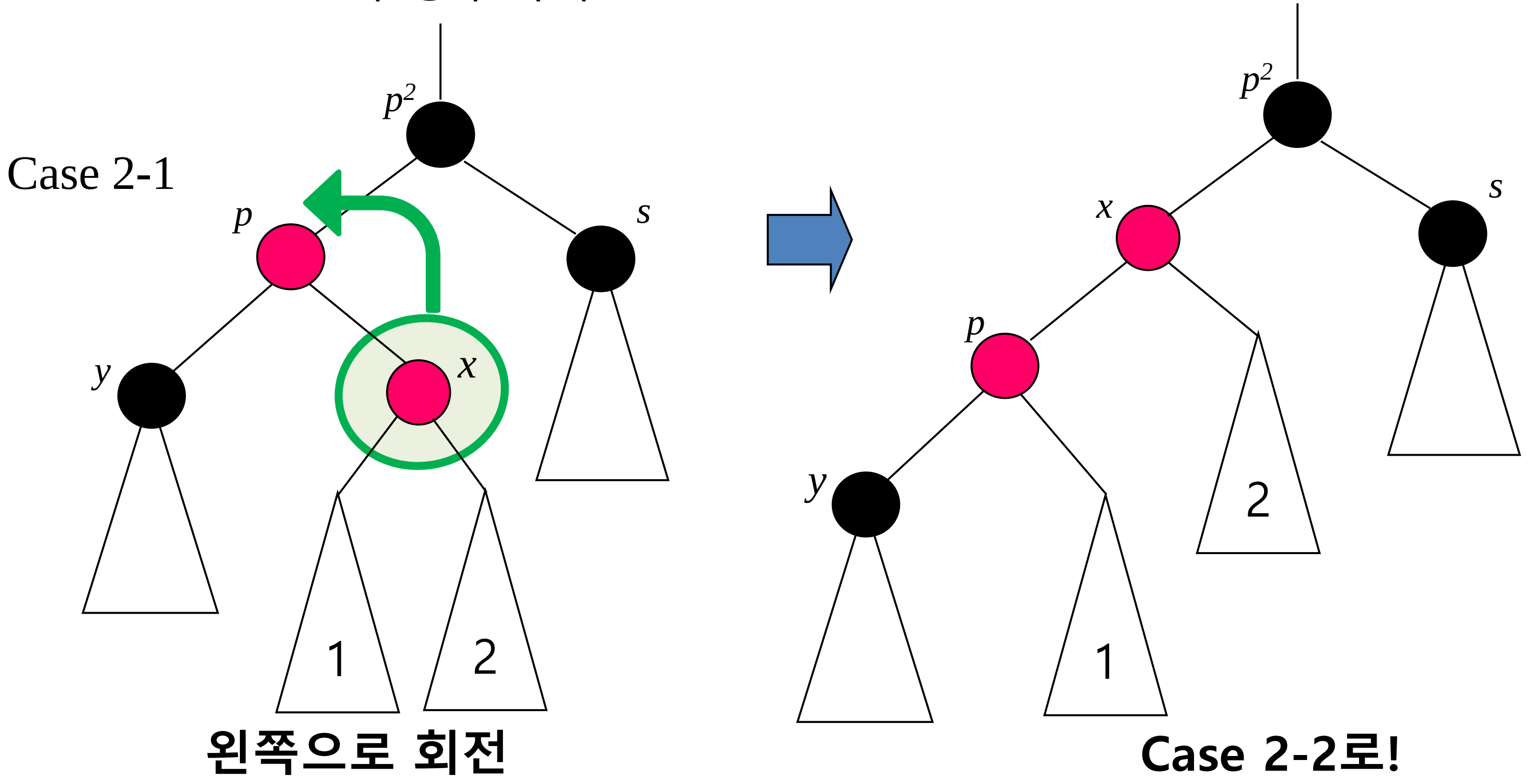
레드블랙트리에서의 삽입

- **Case 1: s가 레드**

- **p와 s(부모와 삼촌)의 색상: 레드 → 블랙**
- **p^2 (할아버지)의 색상: 블랙 → 레드**
  - <u>할아버지가 root 이면,</u>
p^2 가 root 이면, p^2 의 색상을 다시 블랙으로 바꾸고 끝낸다.
  - <u>할아버지에게 조상이 있으면,</u>
p^2 가 root 가 아니면, p^2 의 부모 색상을 확인한다.
    - ✓ p^2 의 부모색상이 블랙이면, 레드 블랙 특성 만족
      - – p^2 : 레드, p^2 부모: 블랙
    - ✓ p^2 의 부모 색상이 레드이면, 레드블랙 특성 ③ 위반,
처음에 x를 삽입한 경우에 발생한 문제가 p^2에 발생한다
      - – 재귀적으로 다시 시작한다.

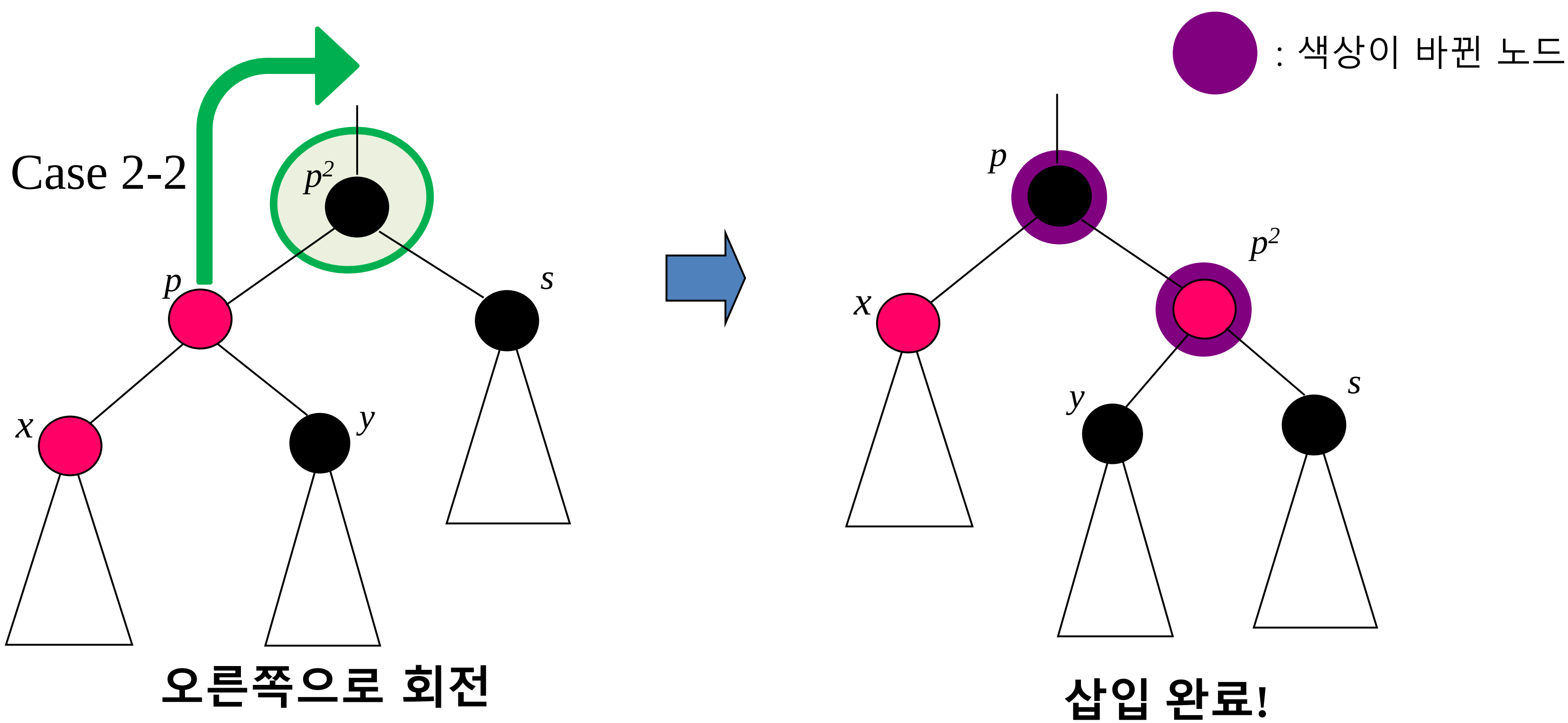
레드블랙트리에서의 삽입

- **Case 2-1: *s*가 블랙이고, *x*가 *p*의 오른쪽 자식**
  - p를 중심으로 왼쪽으로 회전한다
    - p의 자리에 x를 놓고, p는 x의 자식이 된다.
  - 여전히 레드블랙 특성 ③ 위반,
  - Case 2-2 의 경우이다.



레드블랙트리에서의 삽입

- **Case 2-2: *s*가 블랙이고, *x*가 *p*의 왼쪽 자식**
  - p^2 을 중심으로 오른쪽으로 회전한다
  - p와 p^2의 색상을 맞바꾼다.



레드블랙트리에서의 삭제

- 이진 검색 트리에서 삭제와 같다.
- 노드 삭제 후에 색상을 맞추어 준다.

레드블랙트리에서의 삭제

- **임의의 노드 x 삭제**

- **노드 x의 자식이 둘이면**
  - 노드 x의 직후원소(오른쪽 서브트리에서 최소원소)인 노드 m을 노드 x에 복사한 후 노드 m을 삭제한다.
    - ✓ 노드 x의 색상을 변경하지 않고 키값만 바뀌므로, 레드블랙 특성을 위반하지 않는다.
    - ✓ 문제발생은
      - 노드 x의 직후원소 노드m을 삭제한 후, m 주변이 레드블랙 트리 특성을 위반했을 때이다
    - ✓ 최소원소 노드 m은 왼쪽 자식을 갖지 않는다.
      - 즉, 최소원소 노드 m은 최대 하나의 자식만 가질 수 있다.

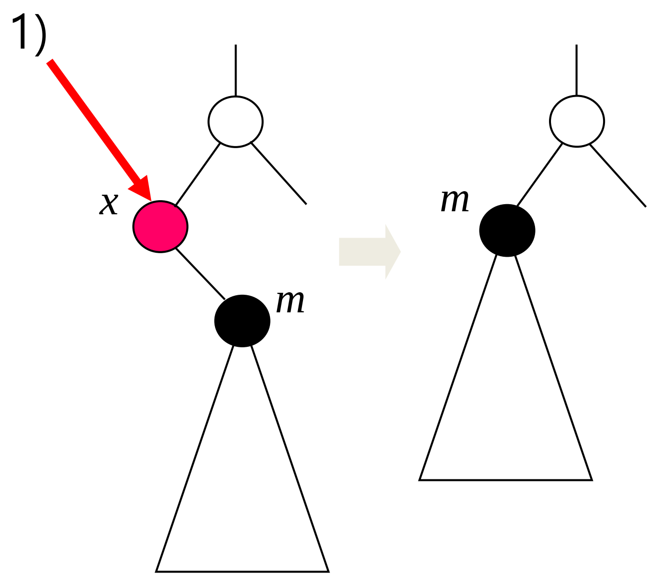
  - **그러므로, 자식이 없거나, 자식이 한 개만 가진 노드의 삭제 작업과 같다.**

레드블랙트리에서의 삭제

- 삭제하려는 노드 x(최대 한 개의 자식을 갖는다)의 자식을 m 이라 가정하자
- x의 자식이 없으면, m은 NIL 노드이다.
- 삭제되는 노드 x는 자기 부모의 왼쪽 또는 오른쪽 자식일 수도 있다. 둘 중 하나만 설명해도 완결성이 떨어지지 않음
- x는 자기 부모의 왼쪽 자식이라고 가정하자

레드블랙트리에서의 삭제

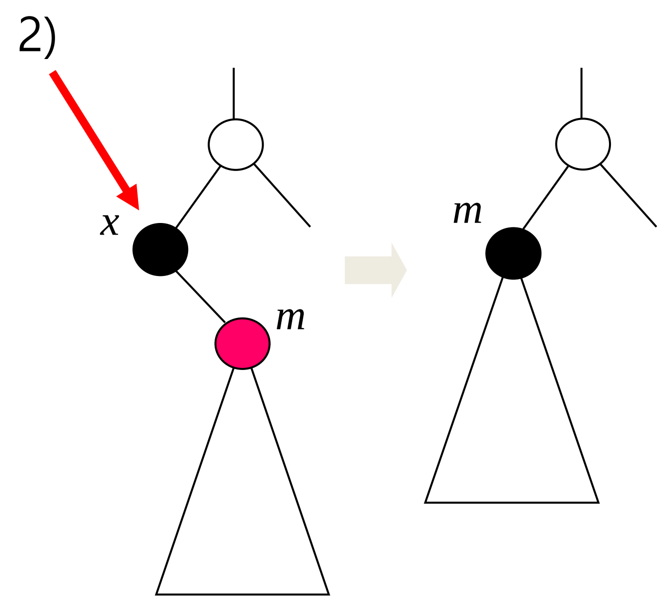
1) x가 레드이면, 삭제후 조치 필요없음.
→ 레드 블랙 특성 깨지지 않음.



레드블랙트리에서의 삭제

- **삭제 노드가 블랙이라도 (유일한) 자식이 레드이면 문제 없다**

- **x가 블랙이고 m이 레드이면,**
  x 삭제(즉, m을 x에 복사)한 후에
  m의 색상을 블랙으로 바꾸면,
  레드블랙 특성을 만족한다.

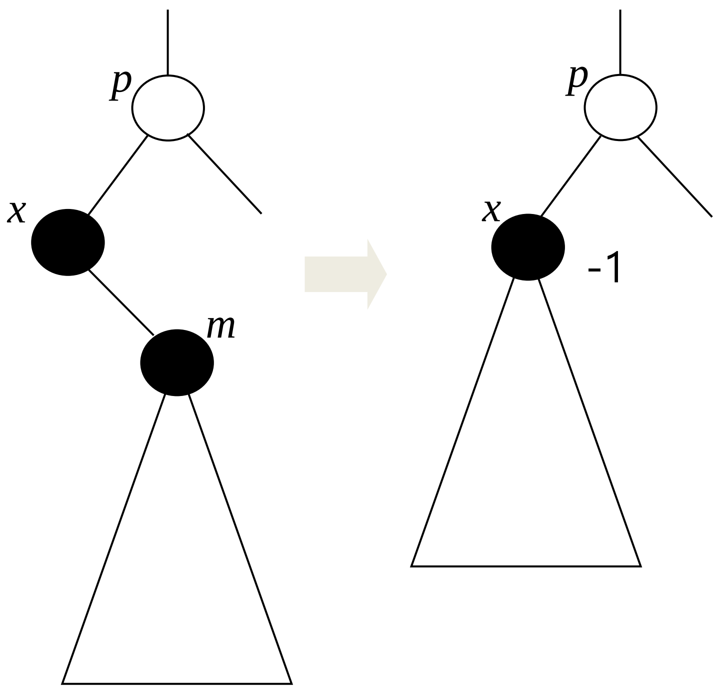


- **삭제노드 x의 부모노드 색상이 표시되지 않은 이유:**
  - 어떤 색상이든 상관없으므로

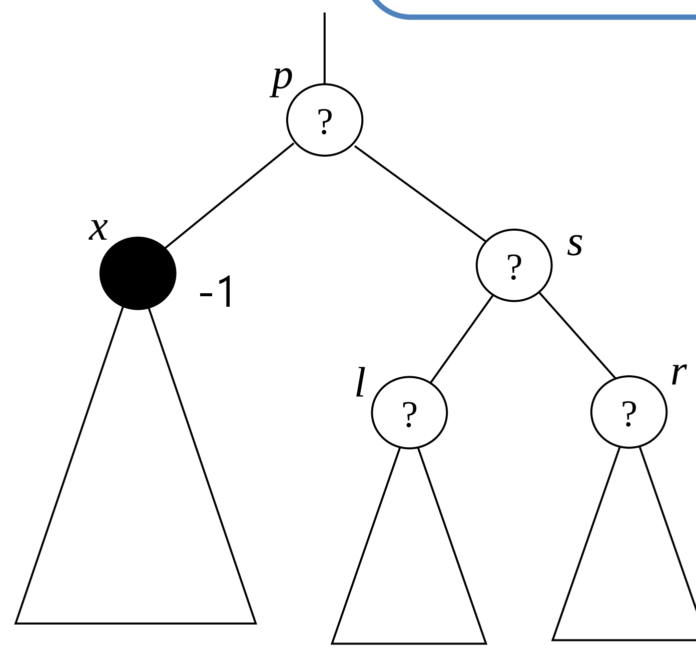
레드블랙트리에서의 삭제

- **x와 m의 색상이 모두 블랙인 경우**

x 옆의 -1은 루트에서
x 를 통해 리프에
이르는 경로에서
블랙 노드의 수가
하나 모자람을 의미함



*x*삭제 후 문제 발생
(레드블랙특성 ④ 위반)



*x*의 주변 상황에 따라
처리 방법이 달라진다

레드블랙트리에서의 삭제

- **x와 m의 색상이 모두 블랙인 경우**

- **Case 1: p가 레드(s는 항상 블랙이므로)<l, r의 색상>에 따라**
  - Case 1-1 <블랙, 블랙>
  - Case 1-2 <*, 레드> → <black, red>, <red, red>
  - Case 1-3 <레드, 블랙>

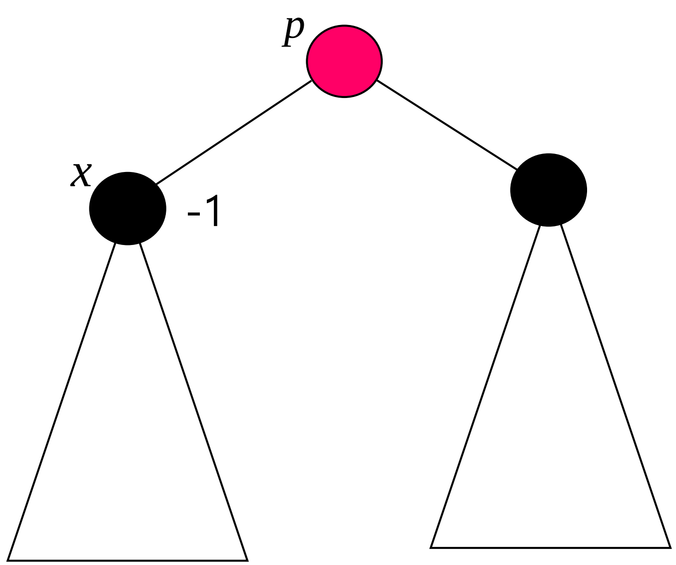
- **Case 2 : p가 블랙 <s, l, r의 색상>에 따라**
  - Case 2-1 <블랙, 블랙, 블랙>
  - Case 2-2 <블랙, *, 레드>
  - Case 2-3 <블렉, 레드, 블랙>
  - Case 2-4 <레드, 블랙, 블랙>

Case 1-2, 2-2 와 Case 1-3, 2-3은 p의 색상이 처리 방법에 영향을 미치지 않으므로 통합하여 표시

레드블랙트리에서의 삭제

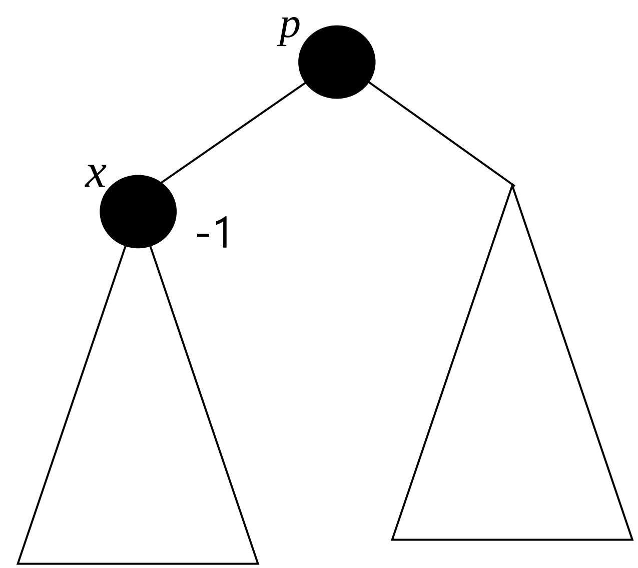
- p의 색상에 따라 경우의 수 나누기

p가 레드



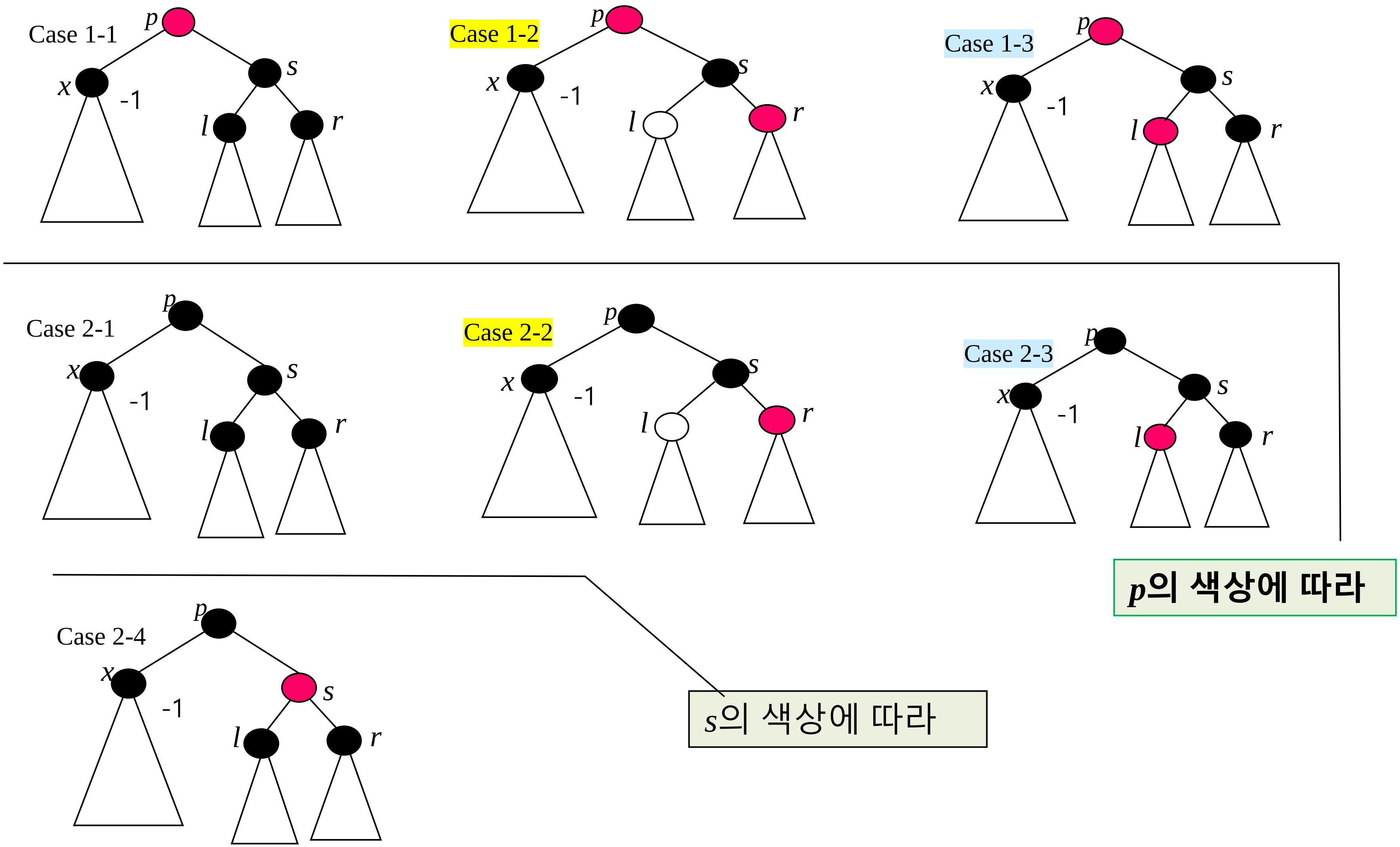
Case 1

p가 블랙



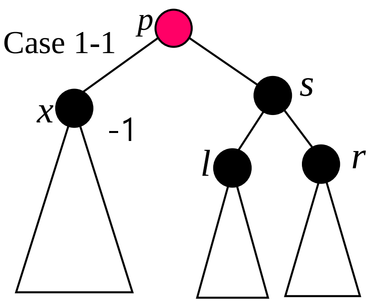
Case 2

레드블랙트리에서의 삭제

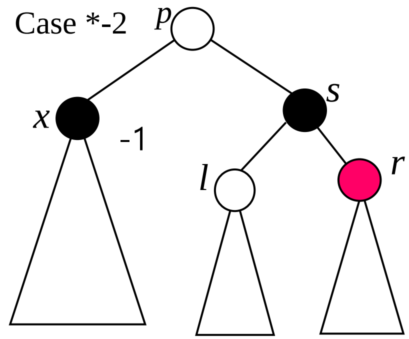


레드블랙트리에서의 삭제

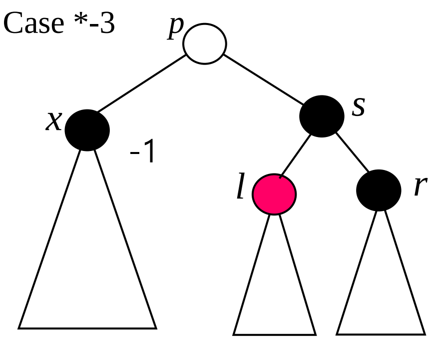
Case 1-1



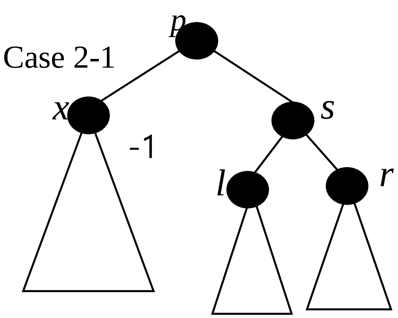
Case *-2



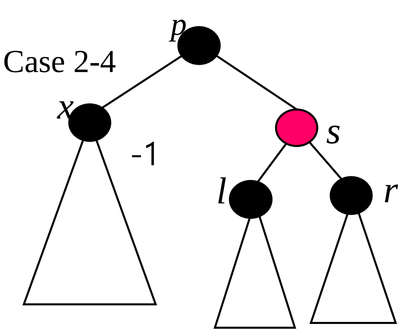
Case *-3



Case 2-1



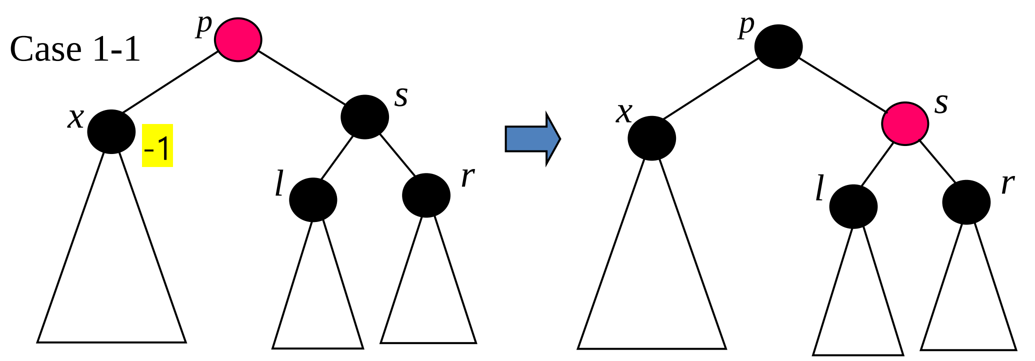
Case 2-4



최종적으로 5가지 경우

레드블랙트리에서의 삭제 ①②③④

- **Case 1- 1:**
  - <l, r의 색상>이 <블랙, 블랙>
  - p와 s의 색상을 맞 바꾼다
    - ✓x에 이르는 경로에서 블랙이 하나 부족한 것이 해결됨.
    - ✓root에서 s를 지나는 경로상의 블랙의 개수는 변환 없음
    - → 특성 ④ 충족됨.



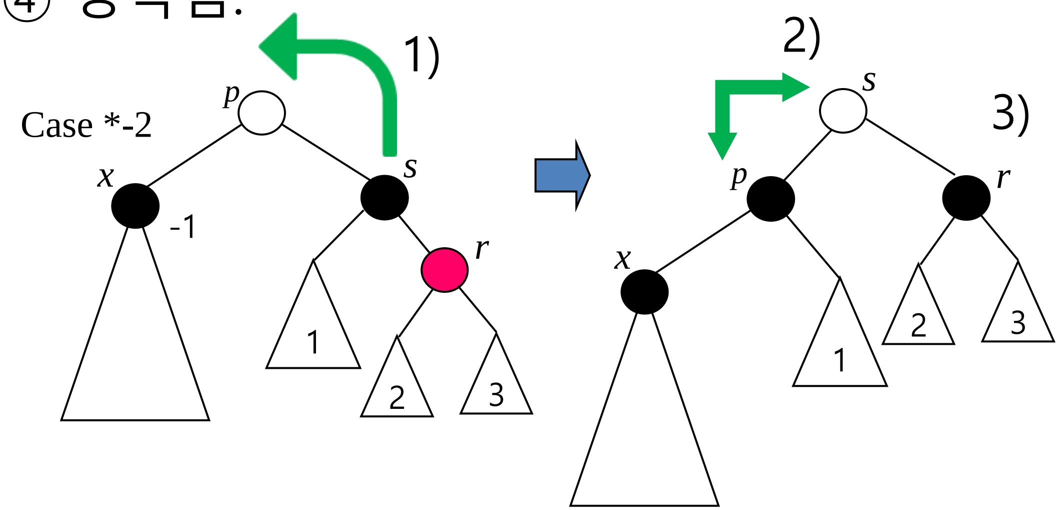
삭제 완료!

레드블랙트리에서의 삭제

- **Case *-2:**
  - Case 1 - 2: <*, 레드>, <l, r의 색상>
  - Case 2 – 2: <블랙, *, 레드>, <s, l, r의 색상>

  1) p를 중심으로 왼쪽으로 회전
  2) p와 s의 색상 바꾼다.
  3) r의 색상을 레드에서 블랙으로 바꾼다.

  → 특성 ④ 충족됨.

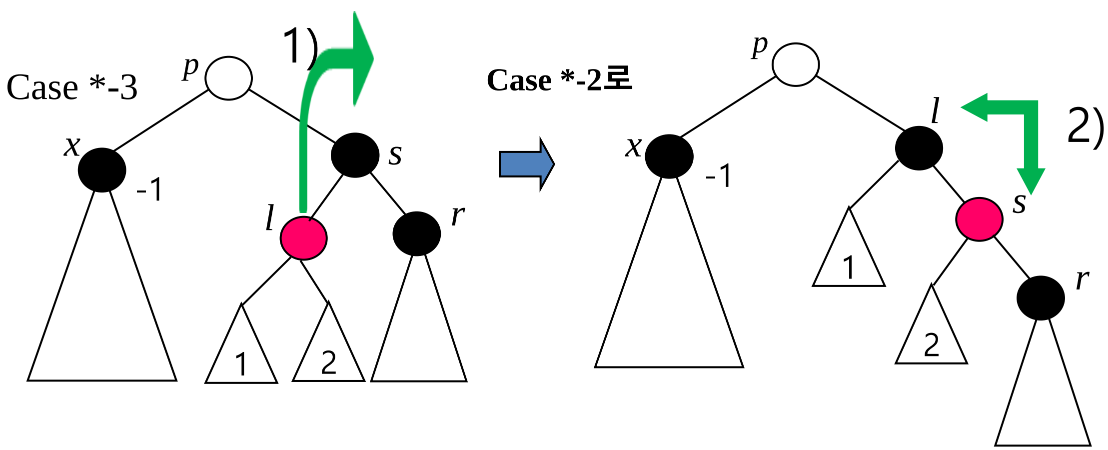


삭제 완료!

레드블랙트리에서의 삭제

- **Case *-3**
  - ➢ Case 1-3 <레드, 블랙>, <l, r의 색상>
  - ➢ Case 2-3 <블렉, 레드, 블랙>, <s, l, r의 색상>

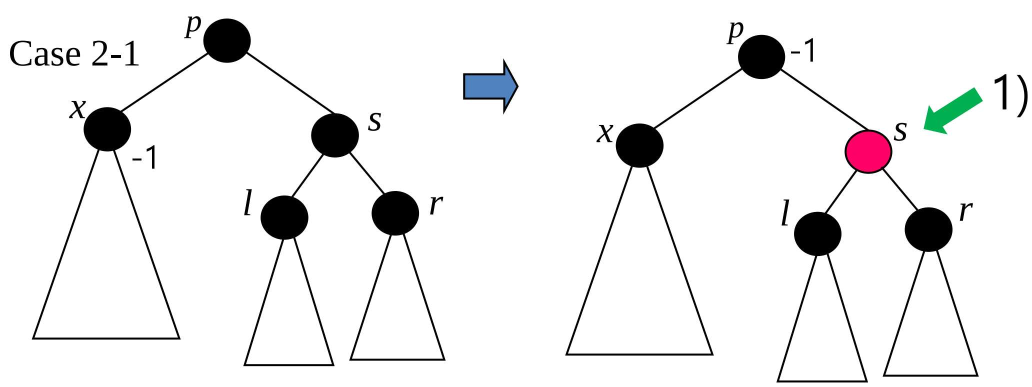
1) s를 중심으로 오른쪽으로 회전한다
2) *l*과 s의 색상을 맞바꾼다.
3) Case * - 2로 이동한다.



레드블랙트리에서의 삭제

- **Case 2 – 1**
  - <블랙, 블랙, 블랙>, <s, l, r의 색상>

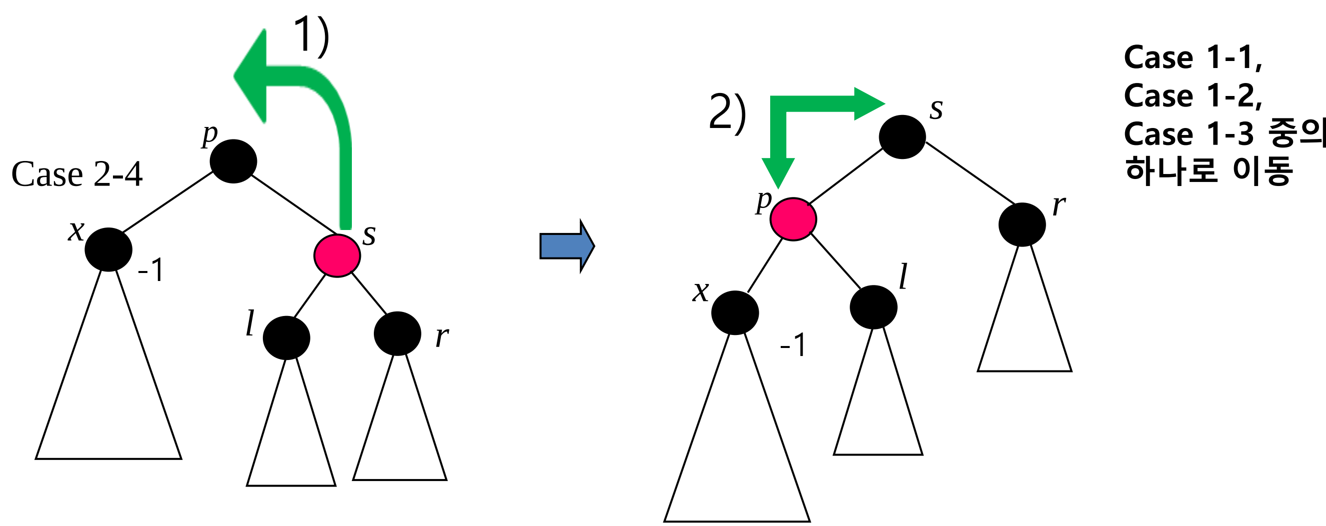
1) s의 색상을 블랙에서 레드로 바꾼다.
2) s를 지나가는 경로에서도 블랙 노드가 한 개 부족하다. p 를 지나는 경로 전체에서 블랙 노드가 한 개 부족한 것이다.
   ✓ 이것은 x에 발생했던 문제가 p에도 발생한 것이다.
3) p를 문제 발생노드로 하여 재귀처리한다.



레드블랙트리에서의 삭제

- **Case 2 – 4:**
  - <레드, 블랙, 블랙>, <s, l, r의 색상>

1) p를 중심으로 왼쪽으로 회전한다
2) p와 s의 색상을 맞 바꾼다.
   ✓ *l*과 r을 경유하는 경로에 문제 발생하지 않는다.
3) Case 1–1, 1-2, 1-3 중 한 경우에 해당되므로, 이동한다.

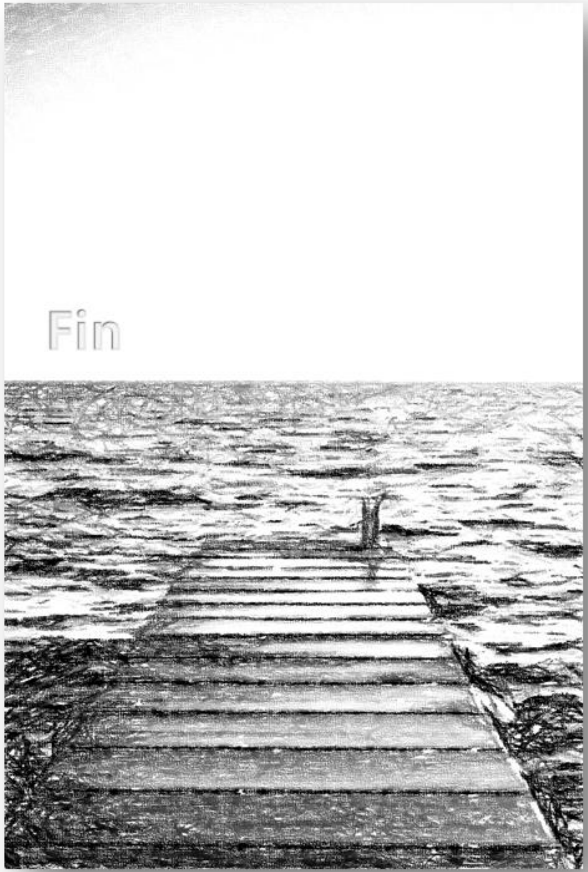


레드블랙트리에서의 검색

- 검색은 트리의 내용을 변경하지 않는다.
- 이진검색트리의 검색과 동일하다.

수행시간 분석

- **트리의 높이: O(log n)**
- **검색: O(log n)**
- **삽입**
  - case 1: case 1 재귀호출. O(log n)
  - case 2: 상수시간.
- **삭제**
  - case 1-1, *-2, *-3: 상수시간
  - case 2-4: case 1-1, 1-2, 1-3으로 이동. 상수시간
  - case 2-1: 재귀호출이 필요할 수 있음. O(log n)
- **∴O(log n)**



https://en.wikipedia.org/wiki/Red-black_tree