# 1 레드 블랙 트리

## 1 레드 블랙 트리

이진 탐색 트리에 균형을 맞추는 기능을 추가한 트리

- 부모 노드보다 작은 값의 노드는 왼쪽, 큰 값의 노드는 오른쪽에 배치됨
- 삽입, 삭제가 일어나는 경우에 높이를 작게 유지하는 자가 균형 이진 탐색 트리

## 1 레드 블랙 트리

- 복잡한 자료구조이지만 실 사용에서 효율적이고
  최악의 경우에도 상당히 우수한 실행 시간을 보임
- 레드 블랙 트리에서는 리프 노드들은 비어있고
  자료를 가지고 있지 않음
- 트리에 n개의 원소가 있을 때 O(log n) 의
  시간복잡도로 삽입, 삭제, 탐색을 할 수 있음

## 1 레드 블랙 트리

- 자료의 삽입, 삭제, 탐색에서 최악의 경우에도 일정한 실행 시간을 보장함
- 실시간 처리와 같은 실행시간이 중요한 경우에 유용함

# 1 레드 블랙 트리

## 순수 이진 탐색 트리에서의 문제점

- 한쪽으로 치우친 사향 이진 트리의 경우
  탐색 효율이 극도로 떨어짐

➡ **[해결책]**
트리가 균형잡힌 모습으로 되도록 하는 알고리즘
레드 블랙 트리는 개략적으로 균형이 잡혀 있음

레드 블랙 트리는 일반적으로 이진 탐색 트리에 비해 효율적

# 1 레드 블랙 트리

## 2 레드 블랙 특성

- 이진 탐색 트리가 가지고 있는 일반적인 조건에
  추가적인 조건을 만족해야 유효한 레드 블랙 트리가 됨

- 이진 탐색 트리의 모든 노드에 블랙 또는 레드의
  색을 칠하되 **레드 블랙 특성**을 만족해야 함

# 1 레드 블랙 트리

## 2 레드 블랙 특성

### 레드 블랙 특성

① 각 노드는 레드나 블랙임

② 루트는 블랙임

③ 모든 리프(NIL 노드)는 블랙임

④ 레드 노드의 자식 노드 양쪽은 언제나 모두 블랙임
(레드 노드는 연속적으로 나타나지 않음 )

⑤ 루트 노드에서 임의의 리프 노드에 이르는 경로에서
만나는 블랙 노드의 수는 모두 같음

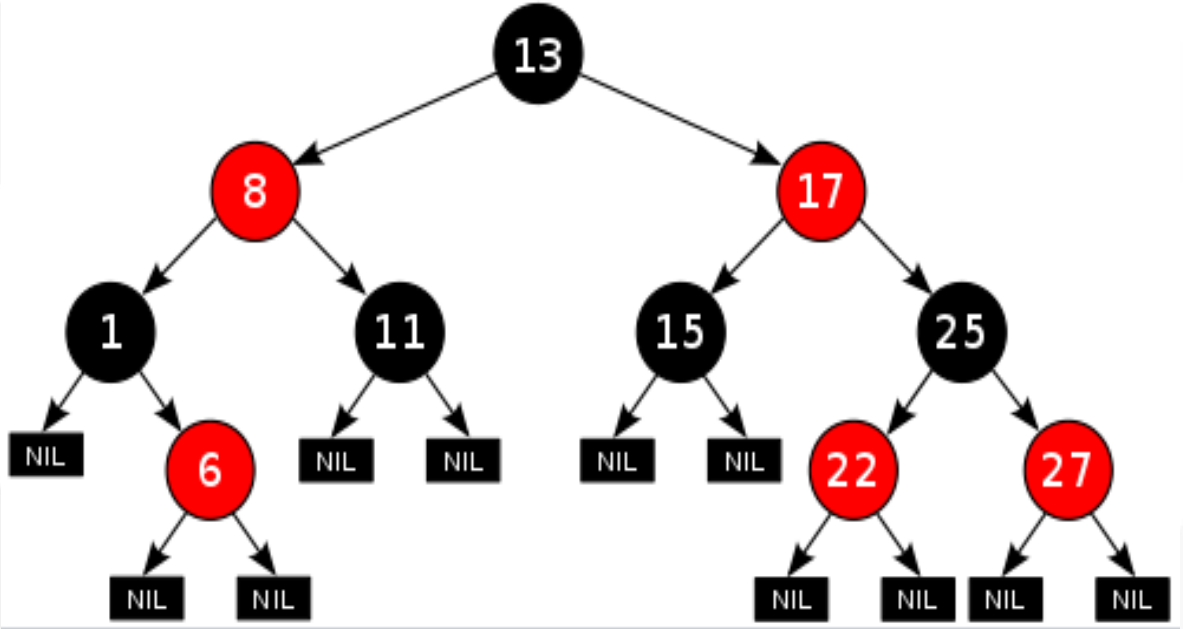
## 2 레드 블랙 특성

### 레드 블랙 특성

- 여기서 리프 노드는 일반적인 의미의
  리프 노드와 다르며 자료를 가지고 있지 않고
  트리의 끝을 나타내는데만 쓰임
- 자식 노드가 존재하지 않을 경우
  NIL 노드라는 특수한 노드가 있다고 가정
- 루트의 부모도 NIL 노드라고 가정

# 1 레드 블랙 트리

## 3 레드 블랙 트리의 예

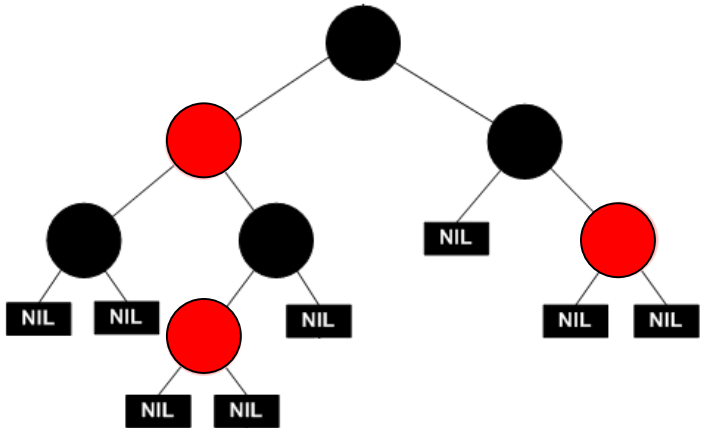



※ 출처 : https://ko.wikipedia.org/wiki/%EB%A0%88%EB%93%9C-%EB%B8%94%EB%9E%99_%ED%8A%B8%EB%A6%AC

# 1 레드 블랙 트리

## 3 레드 블랙 트리의 예

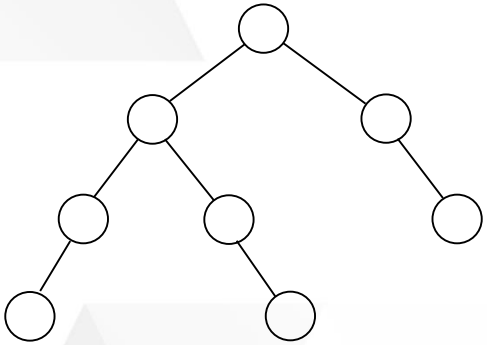
레드 노드는 연속적으로 나타나지 않으나
블랙 노드는 연속으로 나타날 수 있음



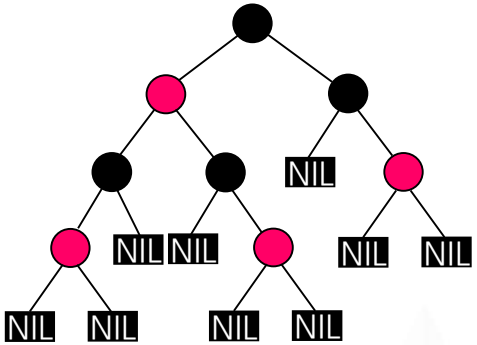

※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 1 레드 블랙 트리

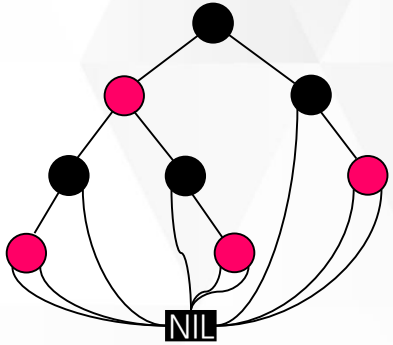
## 4 이진 탐색 트리를 레드 블랙 트리로 만든 예



(a) 이진 탐색 트리의 한 예




(b) (a)를 레드 블랙 트리로 만든 예




(c) 실제 구현시의 NIL 노드 처리 방법

※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

## 5 레드 블랙 트리의 탐색

- 탐색은 트리의 내용을 건드리지 않으므로 이진 탐색 트리에서의 탐색과 동일함

- 삽입과 삭제는 기본적으로는 이진 탐색 트리와 동일하지만 삽입이나 삭제 후 레드 블랙 특성을 위반하는 경우가 발생할 수 있음

  이때 적절한 작업을 해서 레드 블랙 특성을 만족하도록 바로잡아 주어야 함

## 6 레드 블랙 트리에서의 삽입과 삭제

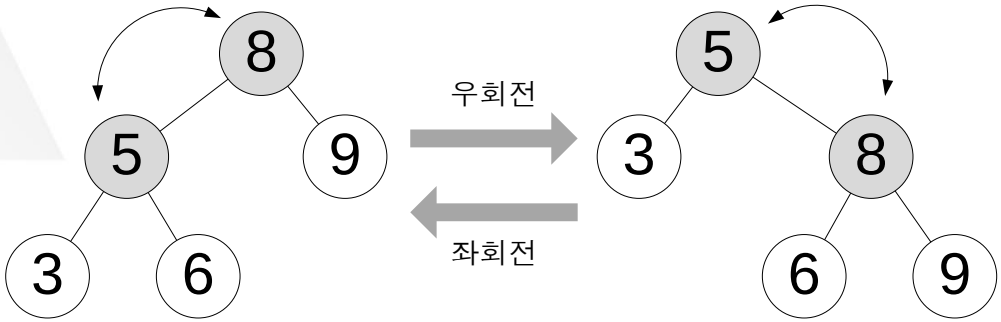
- 삽입이나 삭제 후 레드 블랙 특성을 위반하는 경우 레드 블랙 트리의 특성을 다시 만족하게 하기 위해서 색 변환과 최대 3회의 트리 회전이 필요함 (삽입의 경우 2회)
- 삽입과 삭제는 복잡한 동작이지만 그 복잡도는 여전히 O(log n)임

# 1 레드 블랙 트리

## 7 노드의 회전(Rotation)

- 노드의 회전은 레드 블랙 트리에서 삽입, 삭제 시 공통적으로 필요로 하는 기본 연산임
- 좌회전(Left Rotation), 우회전(Right Rotation)
- 둘 다 이진 탐색 트리에서 한 노드를 중심으로 부분적으로 노드의 모양을 수정하는 방법
- 시간복잡도는 O(1)
- 회전을 하더라도 이진 탐색 트리의 특성을 유지함

## 7 노드의 회전(Rotation)




- 우회전을 할 때에는 왼쪽 자식 노드의 오른쪽 자식 노드를 부모 노드의 왼쪽 자식으로 연결함
- 좌회전을 할 때에는 오른쪽 자식 노드의 왼쪽 자식 노드를 부모 노드의 오른쪽 자식으로 연결함

※ 출처 : 뇌를 자극하는 알고리즘, 박상현, 한빛미디어

# 2 레드 블랙 트리에서 삽입

# 2 레드 블랙 트리에서 삽입

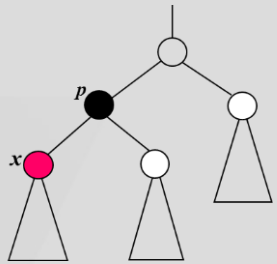
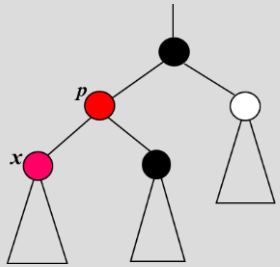
## 1 레드 블랙 트리에서 삽입

- 이진 탐색 트리의 삽입 알고리즘에 따라 삽입한 다음 삽입된 새 노드를 레드로 색칠함
- 다음 단계는 그 주위 노드의 색에 따라 달라짐
- 새 노드는 항상 맨 아래쪽에 매달리므로 삽입 직후에 x의 아래쪽은 블랙 노드인 리프 2개만 있어 레드 블랙 특성에서 문제 생기지 않으므로 x의 위쪽과 관련해서 문제가 생기는지만 확인하면 됨

# 2 레드 블랙 트리에서 삽입

## 1 레드 블랙 트리에서 삽입

### ※ x 노드 삽입

| 만일 x의 부모 노드 p의 색상이 블랙이면 아무 문제 없음(그대로 삽입) | x의 부모 노드 p의 색상이 레드이면 레드 노드가 2개 연속으로 있으므로 레드 블랙 특성이 깨짐 |
| --- | --- |
|  |  노드 색상이 레드-레드인 경우가 문제 |

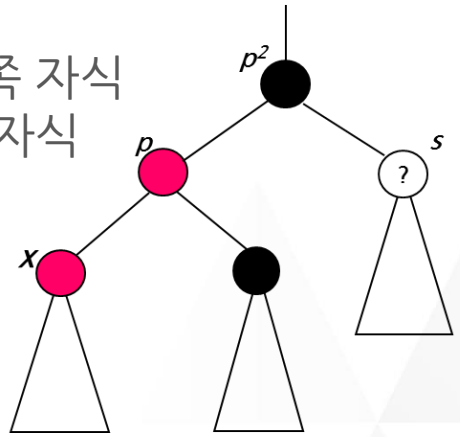
➔ 그러므로 p가 레드인 경우만 고려하면 됨

※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 2 레드 블랙 트리에서 삽입

## 1 레드 블랙 트리에서 삽입

- $p^2$와 x의 형제 노드는 반드시 블랙임
- s의 색상에 따라 두 가지로 나눔
  - Case 1: s가 레드
  - Case 2: s가 블랙
    - Case 2-1: x가 p의 오른쪽 자식
    - Case 2-2: x가 p의 왼쪽 자식



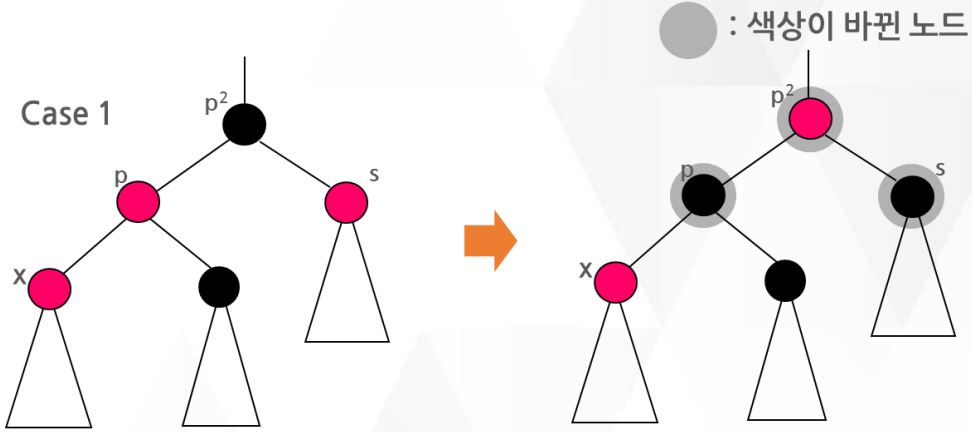

※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 2 레드 블랙 트리에서 삽입

## 1 레드 블랙 트리에서 삽입

### Case 1: s가 레드

- p와 s의 색상을 블랙으로 바꾸고 $p^2$의 색상을 레드로 바꿈
- $p^2$이 루트가 아니면 $p^2$의 부모 색상을 확인해야 함
- $p^2$의 부모가 블랙이면 레드 블랙 특성 모두 만족
- $p^2$의 부모가 레드이면 특성 위반되므로 $p^2$를 문제 발생 노드로 하여 재귀적으로 다시 시작함




✓ $p^2$에서 방금과 같은 문제가 발생할 수 있음

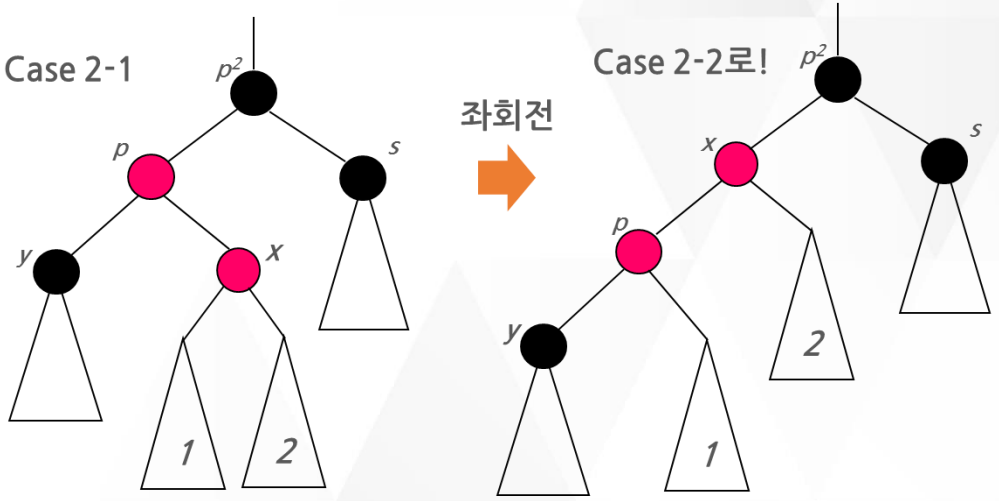
※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 2 레드 블랙 트리에서 삽입

## 1 레드 블랙 트리에서 삽입

### Case 2-1: s가 블랙이고, x가 p의 오른쪽 자식

- p를 중심으로 왼쪽으로 회전함
- 여전히 레드 블랙 특성을 위반함
  ➔ Case 2-2로 이동함

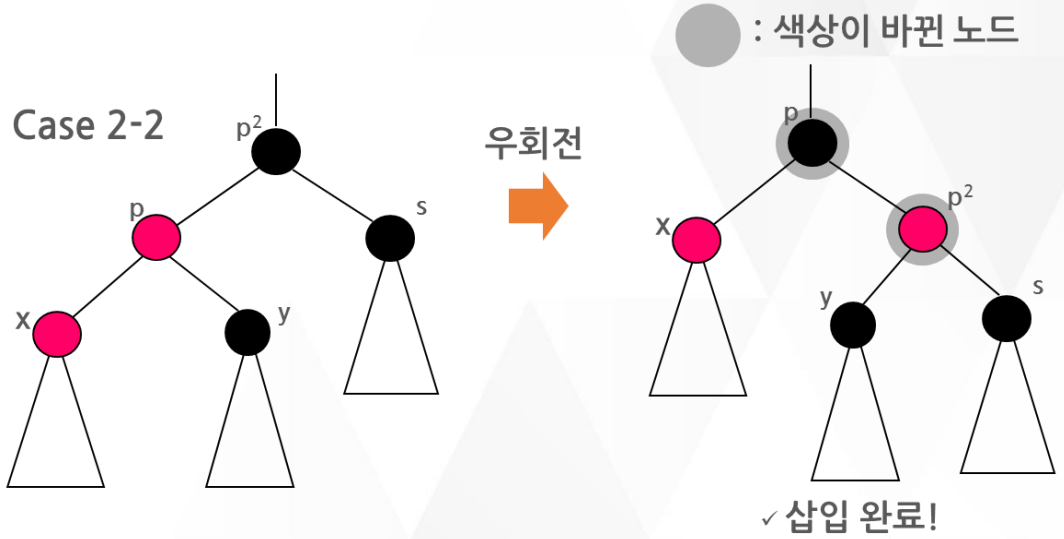



※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

## 1 레드 블랙 트리에서 삽입

### Case 2-2: s가 블랙이고, x가 p의 왼쪽 자식

- $p^2$을 중심으로 오른쪽 회전함
- p와 $p^2$의 색상 바꿈




※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 2 레드 블랙 트리에서 삽입

## 2 레드 블랙 트리에서 삽입 방법

- 트리에서 노드의 색상이 레드-레드인 경우
  이 트리를 부분적으로 조작해서 단번에 해결이
  되거나 혹은 레드-레드 위반이 트리의 위쪽으로
  이동하게 됨
- 위쪽에서 다시 레드-레드 위반을 해결하기 위해
  조작하면 다시 트리의 위쪽으로 이동하게 되는데
  그러다 보면 그 중간쯤에서 자연스럽게
  레드-레드 위반이 해소가 됨

## 2 레드 블랙 트리에서 삽입 방법

- 최악의 경우에는 이것이 루트까지 따라 올라가서 루트가 레드가 되는 상황이 되면 루트를 단순히 블랙으로 바꿔주면 해결됨

- 레드 블랙 트리의 삽입에 대한 시간 복잡도는 트리의 높이를 넘지 않기 때문에 트리의 높이에 비례하는 시간인 O(log n)임

# 3 레드 블랙 트리에서 삭제

# 3 레드 블랙 트리에서 삭제

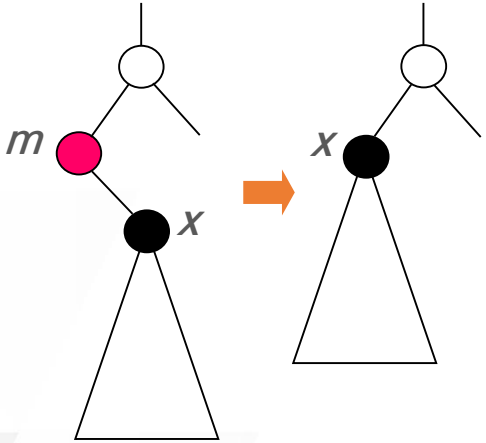
## 1 레드 블랙 트리에서의 삭제

- 이진 탐색 트리의 삭제 방법에 따라 노드를 삭제한 후 색상을 맞춤
- 이진 탐색 트리에서 임의의 노드 d를 삭제할 때 d의 자식이 둘이면 d의 오른쪽 서브 트리에서 최소 원소(d의 직후 원소)를 가진 노드 m의 키를 d로 옮긴 다음 노드 m을 삭제함
- 노드 d의 색상을 건드리지 않고 키만 바뀌는 것은 레드 블랙 특성에 영향을 미치지 않음
- 문제가 되는 것은 최소 원소 m을 삭제한 후 m 주변의 레드 블랙 특성의 위반 여부

# 3 레드 블랙 트리에서 삭제

## 2 레드 블랙 트리에서 삭제가 쉬운 경우

- 삭제 노드를 m이라 가정함
- 삭제 노드가 레드이면 아무 문제 없음
  (삭제후 레드 블랙 특성 깨지 않음)

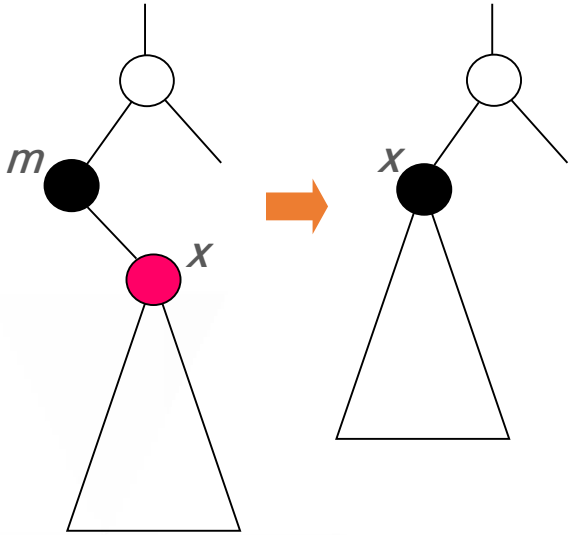



※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 2 레드 블랙 트리에서 삭제가 쉬운 경우

- 삭제 노드가 블랙이라도 (유일한) 자식이 레드이면
  삭제 후 x의 색상을 블랙으로 바꾸면 문제 없음

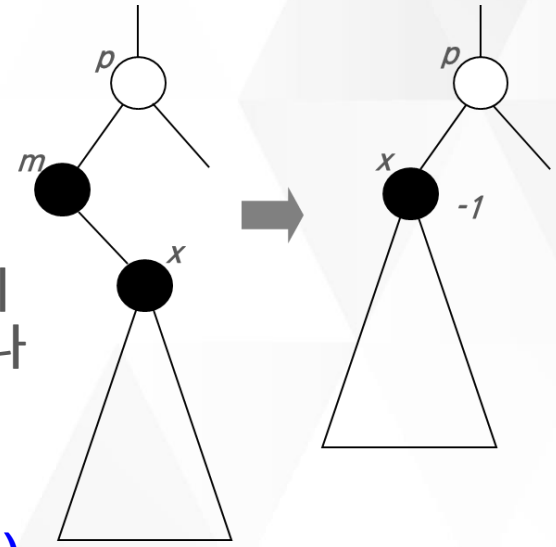



※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 3 레드 블랙 트리에서 삭제가 까다로운 경우

- m과 x의 색상이 블랙일 때가 까다로움
- m이 삭제되면 x는 m의 부모 p의 자식이 되고 루트에서 x를 통과하는 경로의 블랙 노드 개수가 한개 모자라서 레드 블랙 특성이 깨짐




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

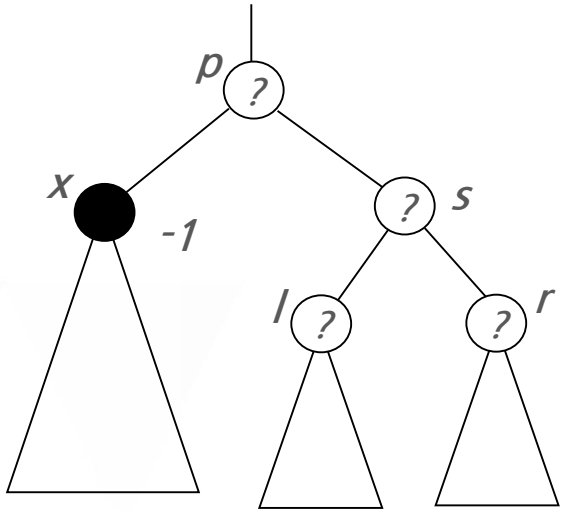
**m 삭제 후 문제 발생 (레드 블랙 특성 위반)**

※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 3 레드 블랙 트리에서 삭제가 까다로운 경우

- x의 주변 노드인 p, s, l, r의 색상 분포에 따라 여러 가지 경우로 나누어 처리함

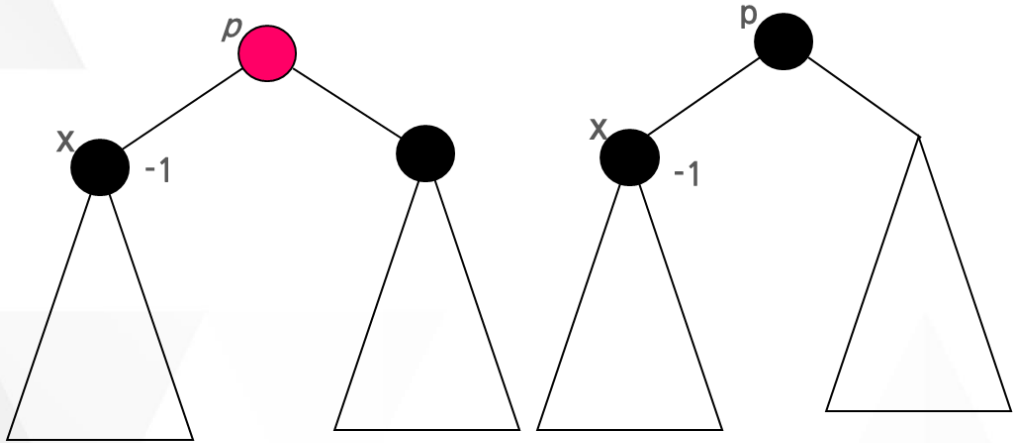



※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 4 경우의 수 나누기

- p의 색상에 따라 Case 1(레드)와 Case 2(블랙)로 나눔



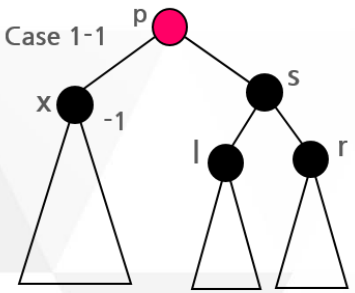

Case 1(p: 레드)　　Case 2(p: 블랙)

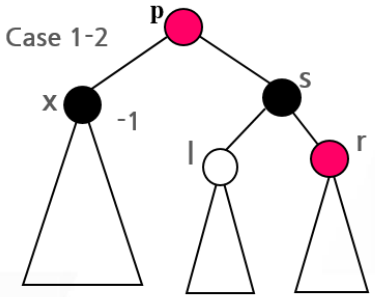
※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

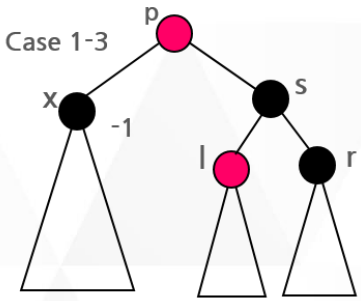
# 3 레드 블랙 트리에서 삭제

## 4 경우의 수 나누기

- Case 1 : p가 레드(s 는 반드시 블랙),
  〈l 의 색상, r 의 색상〉에 따라
  - Case 1-1 〈블랙, 블랙〉
  - Case 1-2 〈레드, 레드〉 또는 〈블랙, 레드〉
    ▷ **〈*, 레드〉로 표시**
  - Case 1-3 〈레드, 블랙〉



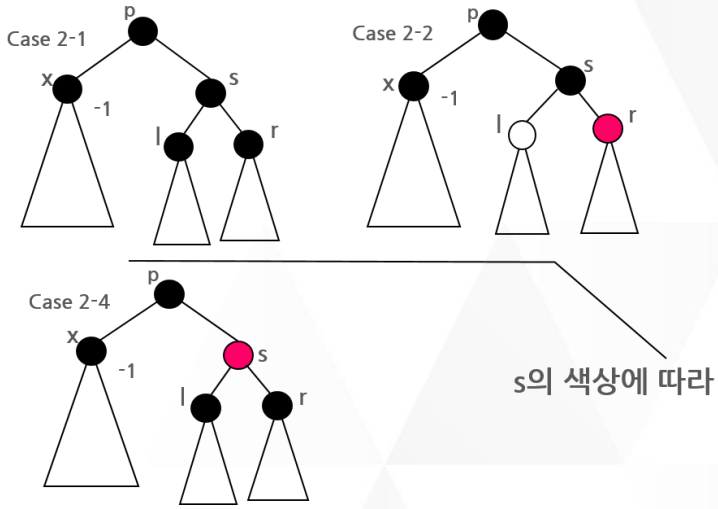



# 3 레드 블랙 트리에서 삭제

## 4 경우의 수 나누기

- p가 블랙이면 s는 블랙, 레드 모두 가능
- Case 2 : p가 블랙 <s의 색상, l의 색상, r의 색상>에 따라
  - Case 2-1 <블랙, 블랙, 블랙>
  - Case 2-2
    <블랙, 레드, 레드> 또는
    <블랙, 블랙, 레드>
    ▷ **<블랙, *, 레드>로 표시**
  - Case 2-3 <블랙, 레드, 블랙>
  - Case 2-4 <레드, 블랙, 블랙>
    ▷ **s가 레드이면**
  - **l 과 r 은 반드시 블랙**



# 3 레드 블랙 트리에서 삭제

## 4 경우의 수 나누기

### 경우의 수 정리

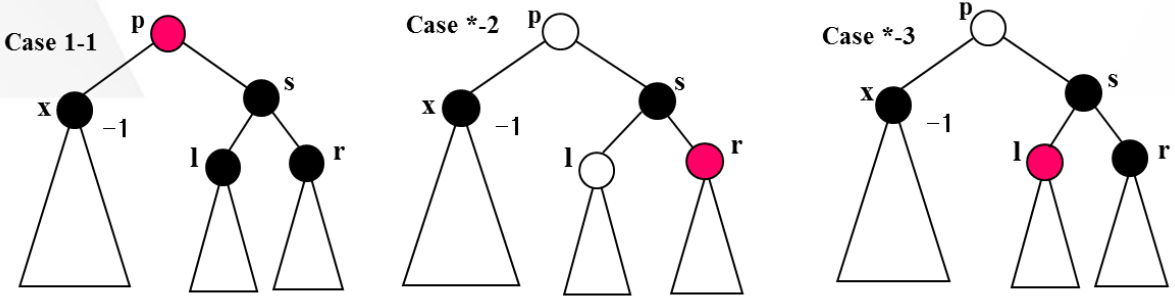
- Case 1-2와 Case 2-2는 p의 색상만 다르고
  p의 색상 처리 방법에 영향을 미치지 않으므로 통합
- Case 1-3과 Case 2-3도 마찬가지 이유로 통합

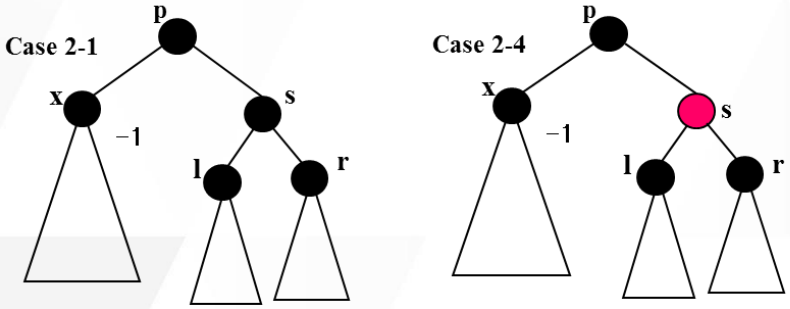
↳ 전체 7가지 경우를 정리하면 5가지인
Case 1-1, Case *-2, Case *-3, Case 2-1,
Case 2-4 로 정리됨

# 3 레드 블랙 트리에서 삭제

## 4 경우의 수 나누기

[레드 블랙 트리에서 삭제를 위한 최종 경우의 수]




✓최종적으로 5가지 경우로 나뉨

※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 5 각 경우에 따른 처리

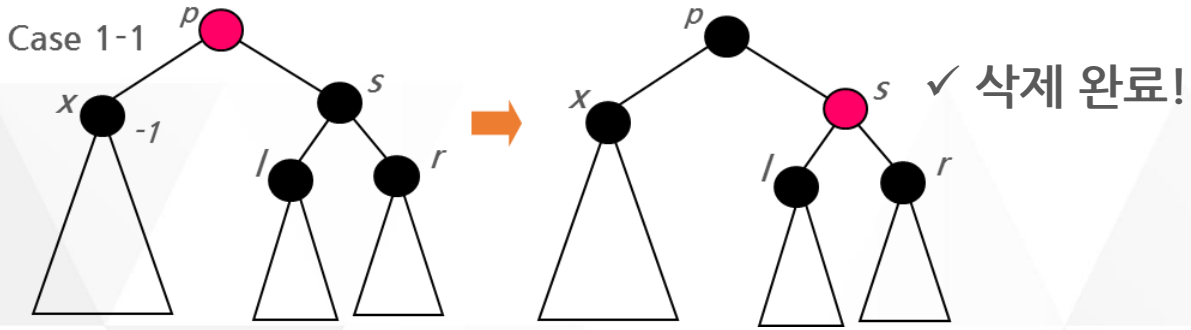
- 5가지의 경우 모두 레드 블랙 트리 특성 중 루트 노드에서 임의의 리프 노드에 이르는 경로에서 만나는 블랙 노드의 수는 모두 같다는 특성을 위반함

# 3 레드 블랙 트리에서 삭제

## 5 각 경우에 따른 처리

### Case 1-1

- 단순히 p와 s의 색상을 맞바꿈
- x에 이르는 경로상에서 블랙이 하나 추가되었으므로
  x에 이르는 블랙 노드가 하나 모자라던 것이 해소됨




✓ 삭제 완료!

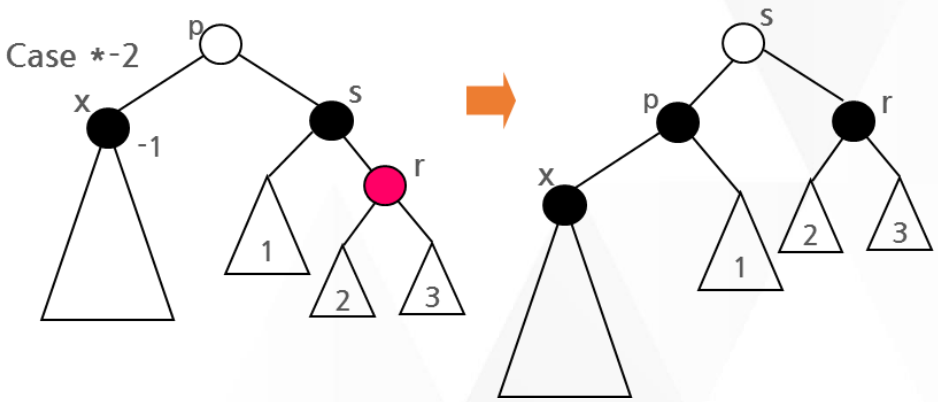
※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 5 각 경우에 따른 처리

### Case *-2

- p를 중심으로 왼쪽으로 회전시키고, p와 s의 색상을 바꾼 다음 r의 색상을 레드에서 블랙으로 바꿈
- x에 이르는 경로상에서 블랙이 하나 추가되어 블랙 노드가 하나 모자라던 것이 해소됨




✓ **삭제 완료!**

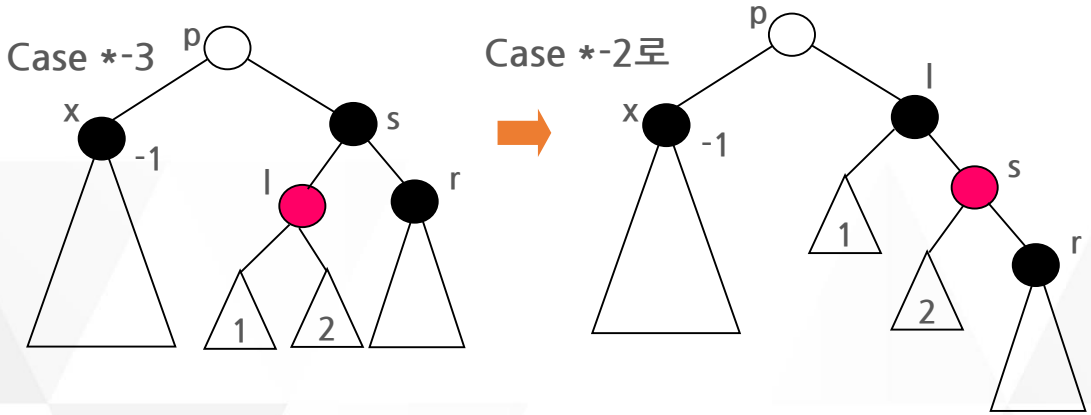
※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 5 각 경우에 따른 처리

### Case *-3

- s를 중심으로 오른쪽으로 회전시키고 l과 s의 색상 바꿈
- Case *-2로 이동함



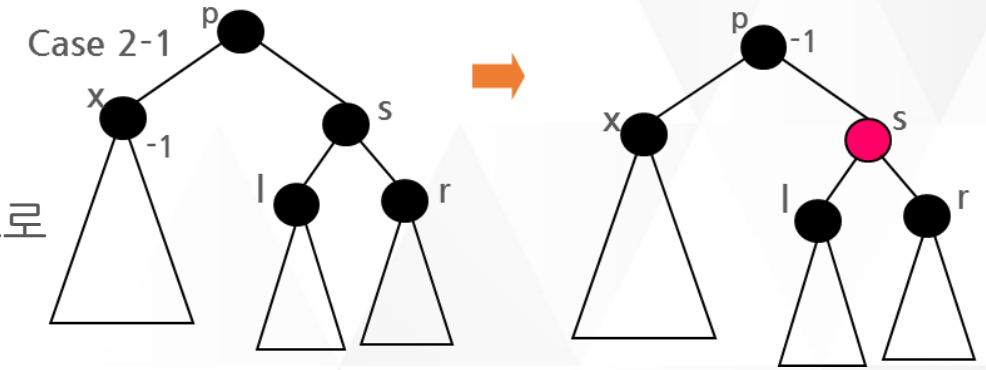

※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 5 각 경우에 따른 처리

### Case 2-1

- 단순히 s의 색상을 블랙에서 레드로 바꿈
- s를 지나가는 경로에서도 블랙 노드가 하나 모자라게 되어 p를 지나가는 경로 전체에서 블랙 노드가 하나 모자라게 됨
- 이것은 원래 x에 대해서 발생했던 문제와 똑같은 문제가 p에 대해 발생했으므로 p를 문제 발생 노드로 하여 재귀적으로 다시 시작함



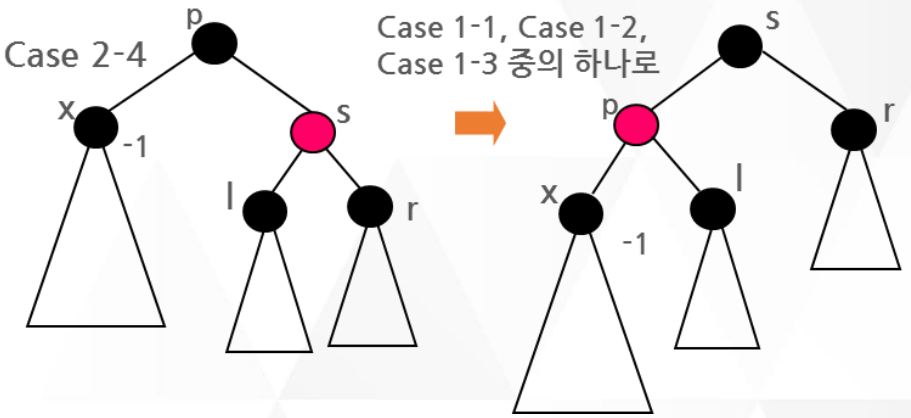

※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 5 각 경우에 따른 처리

### Case 2-4

- p를 중심으로 왼쪽으로 회전시키고 p와 s의 색상 바꿈
- l 과 r 을 경유하는 경로와 관련해서는 문제가 없음
- 문제가 발생한 x의 부모 노드의 색상이 블랙에서 레드로 바뀌었는데 이것은 Case 1에 해당
- 색상 조합에 따라 Case1-1,1-2,1-3 중의 하나로 이동




※ 출처 : 쉽게 배우는 알고리즘, 문병로, 한빛아카데미

# 3 레드 블랙 트리에서 삭제

## 6 레드 블랙 트리의 작업 성능 분석

- 레드 블랙 트리에서 탐색에 소요되는 시간은 O(log n)임
- 삽입과 삭제도 O(log n)