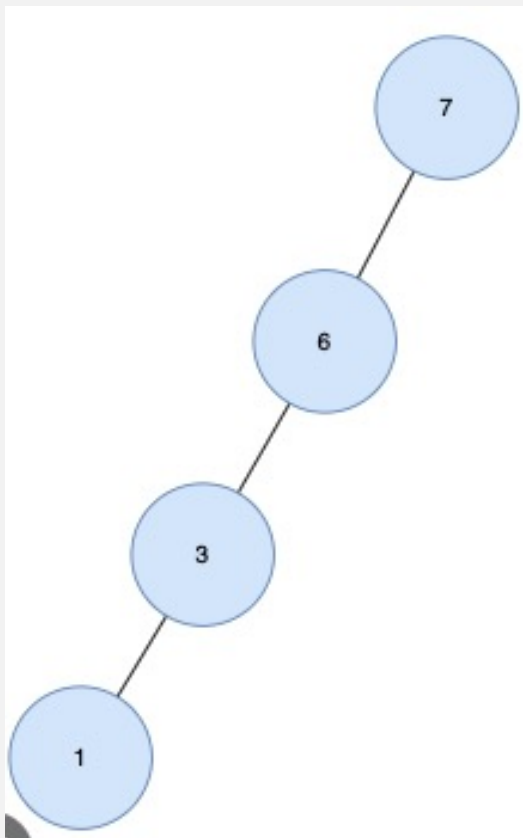


# RED-BLACK TREE

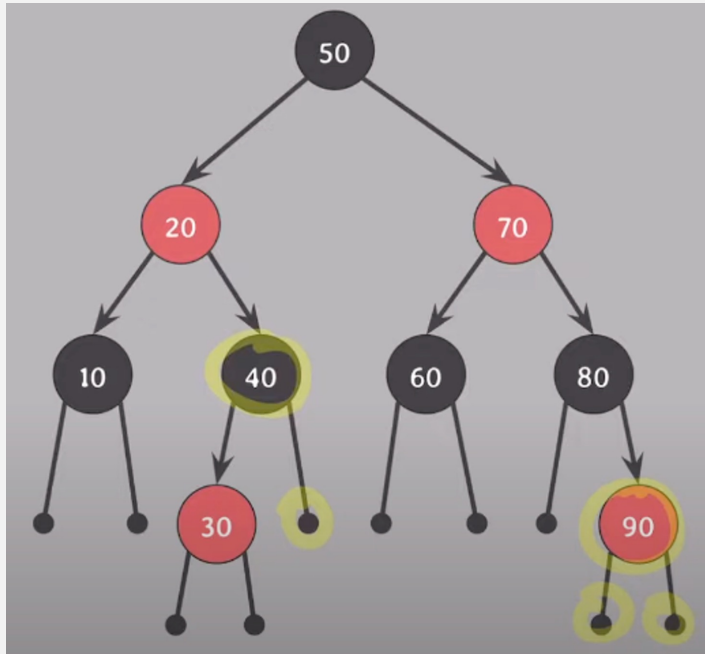
# RED-BLACK TREE

- 이진 탐색 트리의 한 종류 : 각 노드에서 왼쪽에는 작은, 오른쪽에는 큰 값
- 스스로 균형을 잡는 트리
- BST의 worst case의 단점을 개선
  - 최악의 경우 한쪽으로 편향될 수 있다.
  - 모든 노드들을 한 번씩 다 돌아야 값을 찾을 수 있기 때문에 시간복잡도가  $O(N)$
  - 따라서 red-black 트리는 스스로 균형을 맞춰가면서 편향되지 않게 함.  $O(\log N)$
- 모든 노드는 red or black

# BST

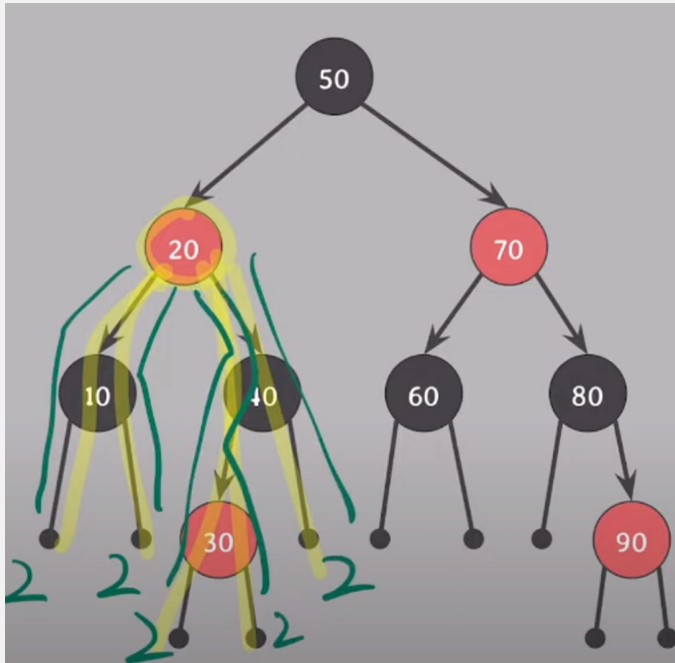


# RED-BLACK TREE 속성



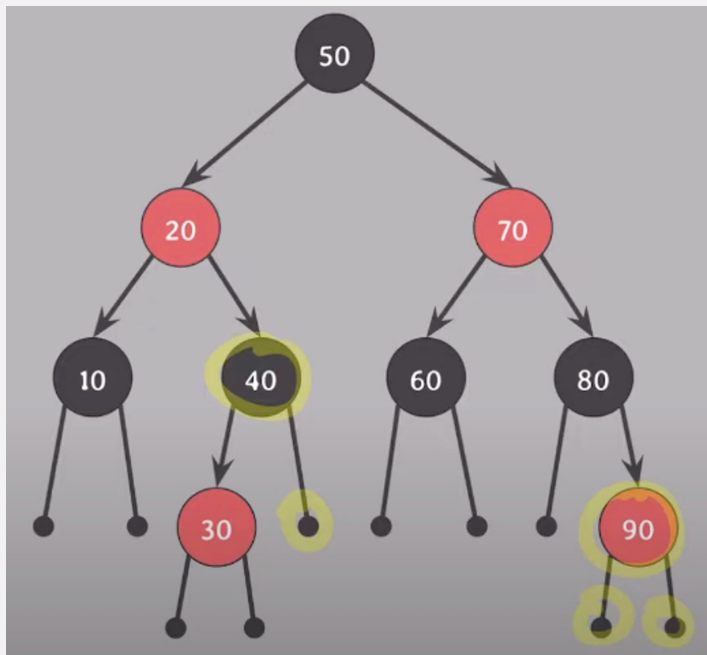
- 1. 모든 노드는 red or black
- 2. root-node는 black
- 3. 모든 null(leaf)노드는 black
- 4. red의 자녀들은 black임. red가 연속적으로 존재할 수 없다.
- 5. 임의의 노드에서 자손 null노드까지 가는 경로들의 black 수는 같다.
  - (자기 자신은 카운트에서 제외) -> 노드의 black height

## RED-BLACK TREE 속성



- 노드 x의 black height
  - 노드 x에서 임의의 자손 nil노드까지 내려가는 경로에서 black 수
  - (자신은 카운트에서 제외)
  - 5번 속성을 만족해야 성립하는 개념

# RED-BLACK TREE 속성



- 1. 모든 노드는 red or black
- 2. root-node는 black
- 3. 모든 null(leaf)노드는 black
- 4. red의 자녀들은 black임. red가 연속적으로 존재할 수 없다.
- 5. 임의의 노드에서 자손 null노드까지 가는 경로들의 black 수는 같다.
  - (자기 자신은 카운트에서 제외) -> 노드의 black height

## RED-BLACK TREE 속성

- 색을 바꾸면서도 5번 속성을 유지하기
  - RB tree가 5번 속성을 만족하고 있고 두 자녀가 같은 색을 가질 때 부모와 두 자녀의 색을 바꿔줘도 5번 속성은 여전히 만족한다.

# 블랙 노드의 두 번 연속은 가능?

 You

아 색을 변경하는 것은 과정 중에 하나로 수행되는 거고 사실을 연속된 색이 두 번 나오면 안되는 건 맞다는거지? 할아버지 노드 블랙, 부모노드 블랙, 자손노드 레드 인 상태도 어쨌든 조정이 필요하다는 거지?

 ChatGPT

맞습니다. 레드-블랙 트리에서 색상 변경은 트리의 균형을 유지하는 과정 중 하나로 수행되며, 레드 노드가 연속해서 나오는 것은 허용되지 않습니다. 즉, 레드 노드의 부모 노드는 반드시 블랙이어야 합니다. 이는 레드-블랙 트리의 기본 규칙 중 하나입니다.

할아버지 노드가 블랙, 부모 노드가 블랙, 자손 노드가 레드인 상태는 기본적으로 규칙을 위반하지는 않습니다. 하지만, 특정 연산(삽입 또는 삭제)을 수행하는 과정에서 이러한 구성이 변경되어 레드 노드가 연속으로 나타나게 될 경우, 트리의 균형을 유지하기 위한 추가적인 조정이 필요합니다.

예를 들어, 레드 노드가 삽입되거나 삭제된 후에 레드-블랙 트리의 균형이 깨진다면, 여러 조정 기법이 사용될 수 있습니다. 이러한 조정 기법에는 색상 변경과 회전이 포함되며, 이들은 트리의 속성을 만족시키면서도 효율적인 탐색, 삽입, 삭제 연산을 보장하기 위해 사용됩니다.

따라서, 레드-블랙 트리에서의 색상 변경과 회전은 트리의 구조적 균형을 유지하고 효율적인 연산을 보장하기 위해 필수적인 부분입니다.

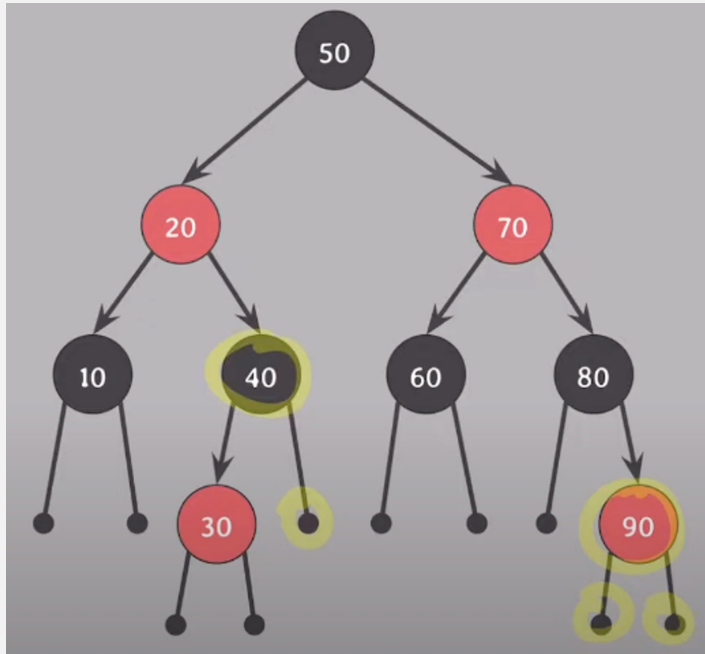
   



## RB TREE는 어떻게 균형을 잡는가?

- 삽입/삭제 시 주로 4번, 5번 속성을 위반하며, 이들을 해결하려고 구조를 바꾸다 보면 자연스럽게 트리 균형이 잡히게 된다.

# RED-BLACK TREE 속성

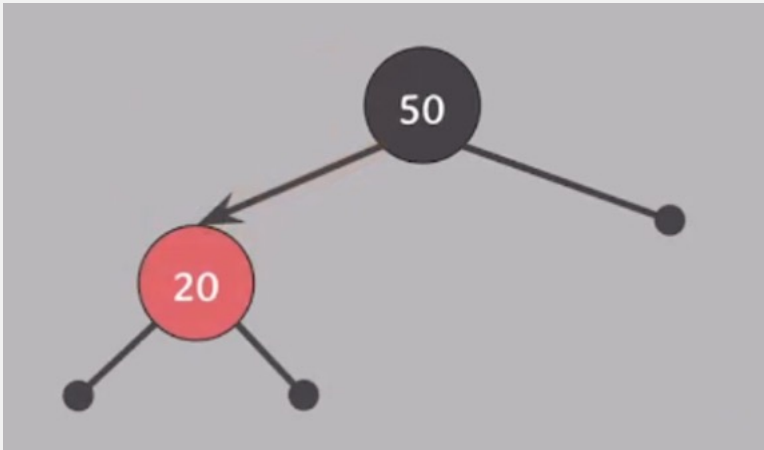


- 1. 모든 노드는 red or black
- 2. root-node는 black
- 3. 모든 null(leaf)노드는 black
- 4. red의 자녀들은 black임. red가 연속적으로 존재할 수 없다.
- 5. 임의의 노드에서 자손 null노드까지 가는 경로들의 black 수는 같다.
  - (자기 자신은 카운트에서 제외) -> 노드의 black height

## RB TREE 삽입 방식

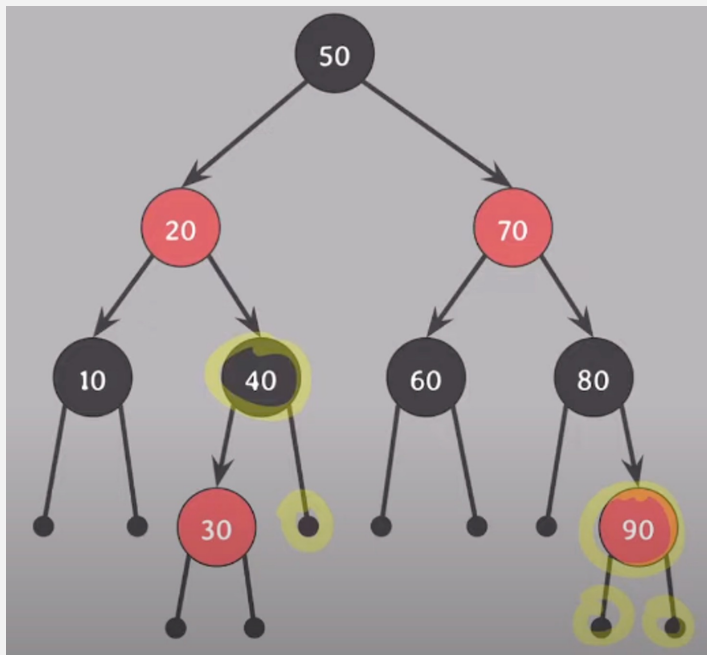
- 0. 삽입 전 RB Tree 속성 만족한 상태
- 1. 삽입 방식은 일반적인 BST와 동일
- 2. 삽입 후 RB트리 위한 속성 여부 확인
- 3. 삽입 후 RB트리 속성을 위반했다면 재조정
- 4. RB트리 속성을 다시 만족

# INSERT



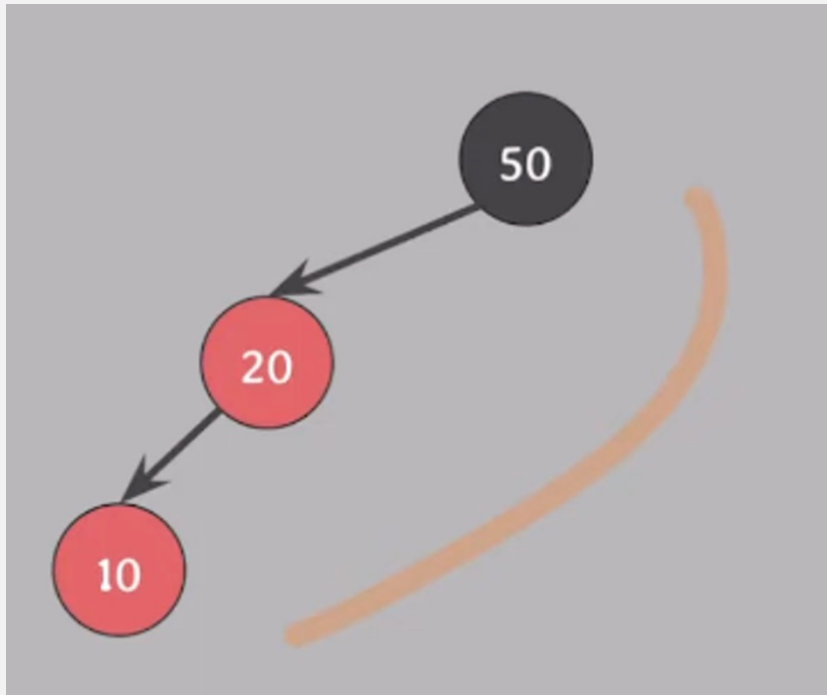
- 삽입하는 노드는 **항상 RED**
- Insert(50)
- **Insert(20)**
- 노드를 삽입할 때 두 null노드의 색은 black으로 고정한다.
- 그러면 자연스럽게 3번 속성을 만족

## RED-BLACK TREE 속성



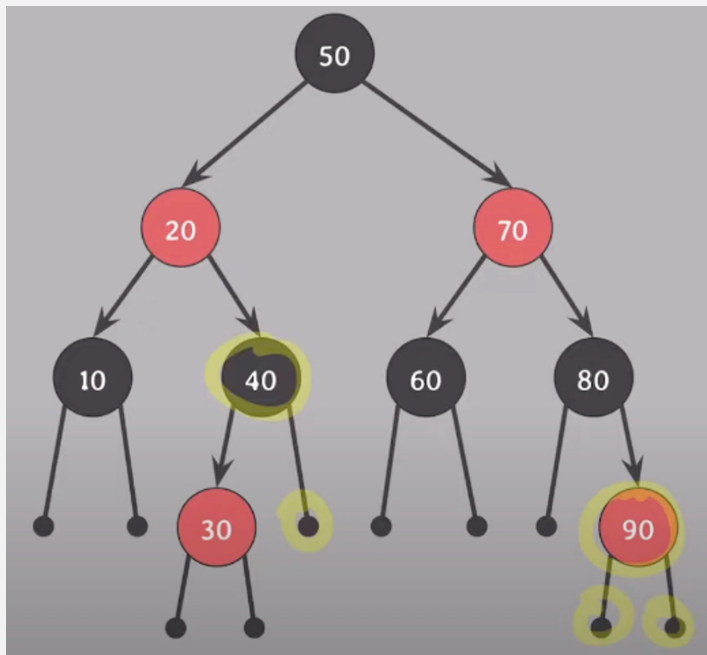
- 1. 모든 노드는 red or black
- 2. root-node는 black
- 3. 모든 null(leaf)노드는 black
- 4. red의 자녀들은 black임. red가 연속적으로 존재할 수 없다.
- 5. 임의의 노드에서 자손 null노드까지 가는 경로들의 black 수는 같다.
  - (자기 자신은 카운트에서 제외) -> 노드의 black height

# INSERT



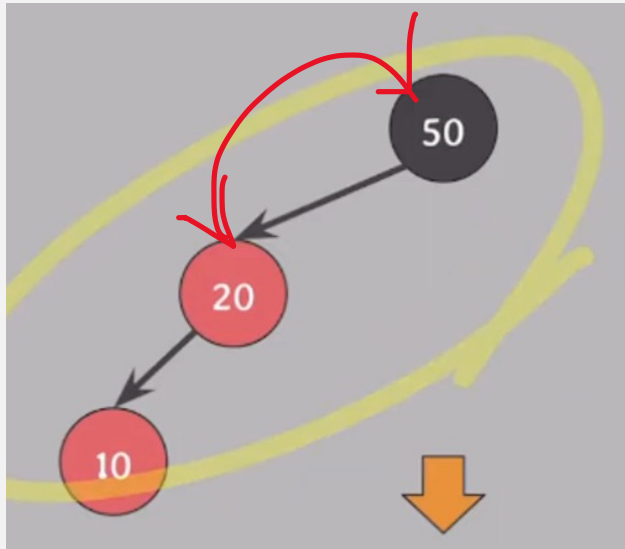
- Insert (10)
- 4번 속성 위반
  - 어떻게 해결할 수 있을까?

# RED-BLACK TREE 속성



- 1. 모든 노드는 red or black
- 2. root-node는 black
- 3. 모든 null(leaf)노드는 black
- 4. red의 자녀들은 black임. red가 연속적으로 존재할 수 없다.
- 5. 임의의 노드에서 자손 null노드까지 가는 경로들의 black 수는 같다.
  - (자기 자신은 카운트에서 제외) -> 노드의 black height

# INSERT

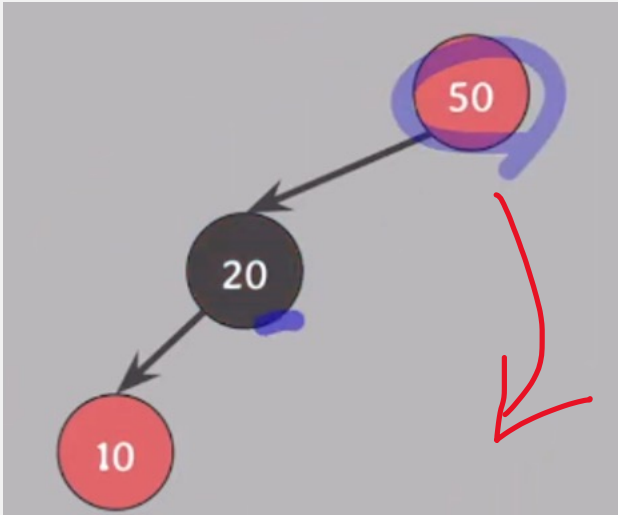


- 해결방법
  - Red가 한 쪽으로 몰려 있으니 red 하나를 반대편으로 옮겨준다면?
  - 하지만 잊지 말아야 할 것은 구조를 바꿨을 뒤에도 BST 특징 또한 유지가 되어야 한다.

<- BST 위반! 따라서 회전을 시키기!

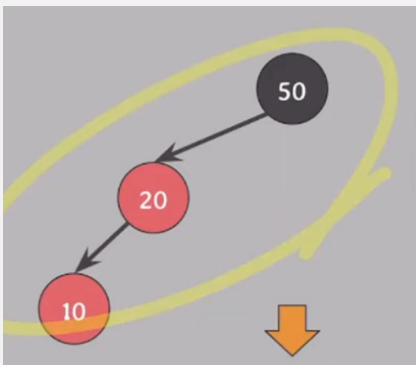


# INSERT

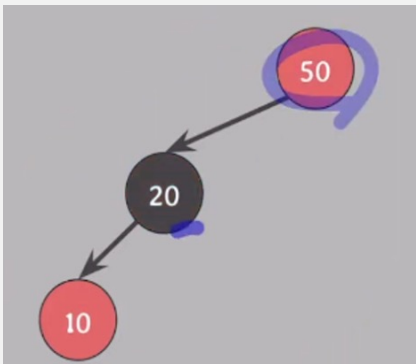


- 1. 20과 50의 색을 바꿔준다.
- 2. 50을 기준으로 오른쪽으로 회전한다.
- => 4번 속성을 포함해서 RB Tree 만족

## CASE.3 해결

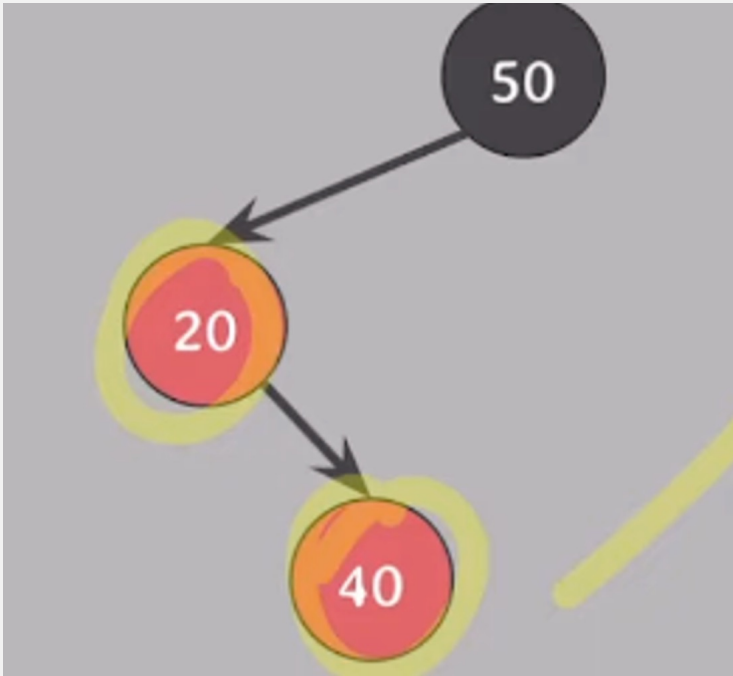


- Red 삽입 후 4번 속성을 위반했을 때
- 삽입된 red노드가 부모의 왼쪽
- & 부모도 red고 할아버지의 왼쪽 자녀
- & 삼촌은 black 이라면



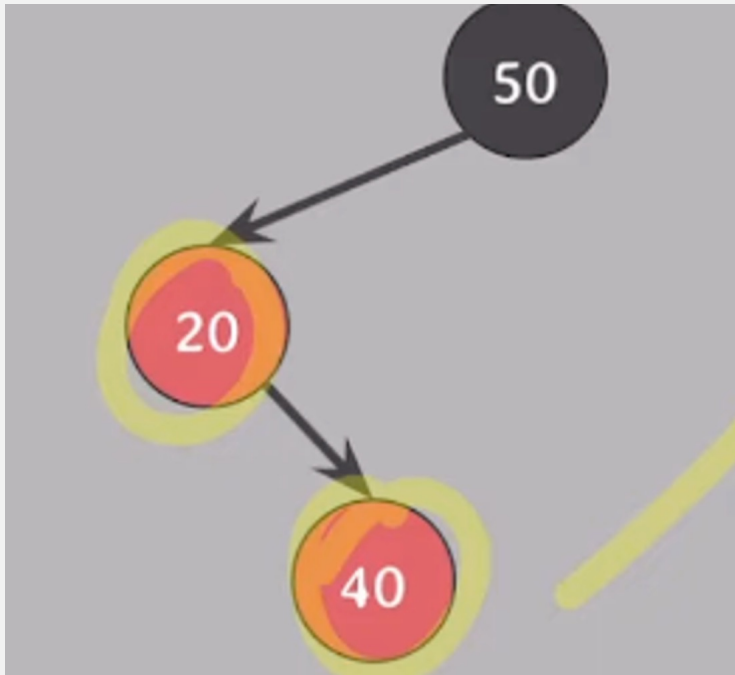
- 부모와 할아버지 색을 바꾼 후 할아버지를 기준으로 오른쪽으로 회전한다.
- # 왼쪽 오른쪽 바뀌도 성립

## 다른 케이스(INSERT)



- 삽입하는 노드는 항상 RED
- Insert(40)
- 4번 속성 위반
  - 이유 : 노드가 red라면 자녀들은 black
- 즉, red가 몰려있음.

## 다른 케이스(INSERT)

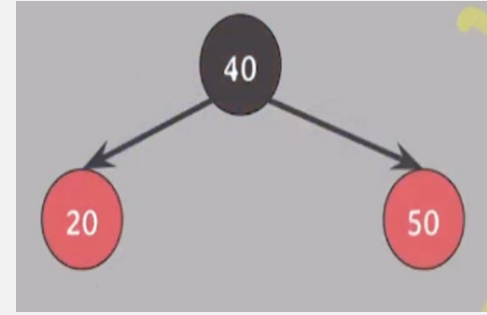
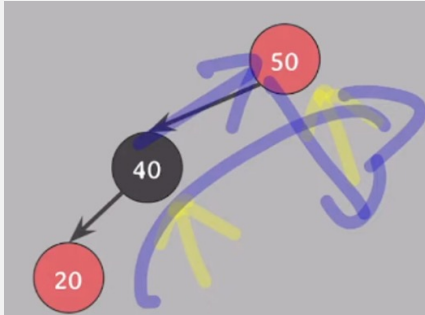
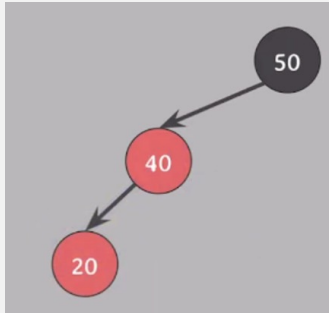


- 4번 속성 위반을 해결하기 위해 red하나를 넘겨야 하는데 BST 특징 또한 유지하면서 넘기려면 회전을 사용해야함.
- 회전을 어떻게 할 것인가?

## 다른 케이스(INSERT)

- Case.3 과 다른 점은 할아버지까지 가는 **경로가 꺾였다는 점.**
- 그럼 어떻게 ?
- 꺾인 부분은 펴줘서 case.3과 같은 형태로 만들면 case.3과 같은 방식으로 해결 가능
- 회전 후에도 4번 이외의 속성들을 만족하며 이제 case.3의 형태가 됨.

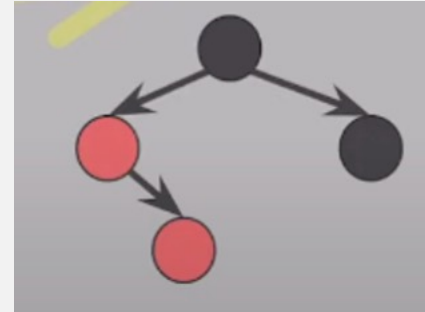
## 다른 케이스(INSERT)



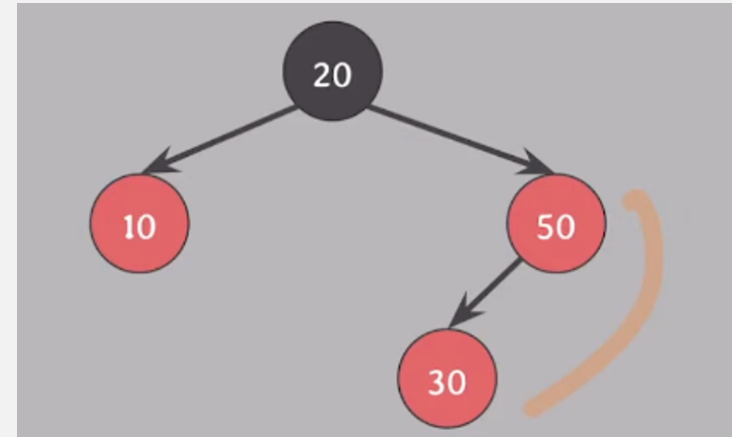
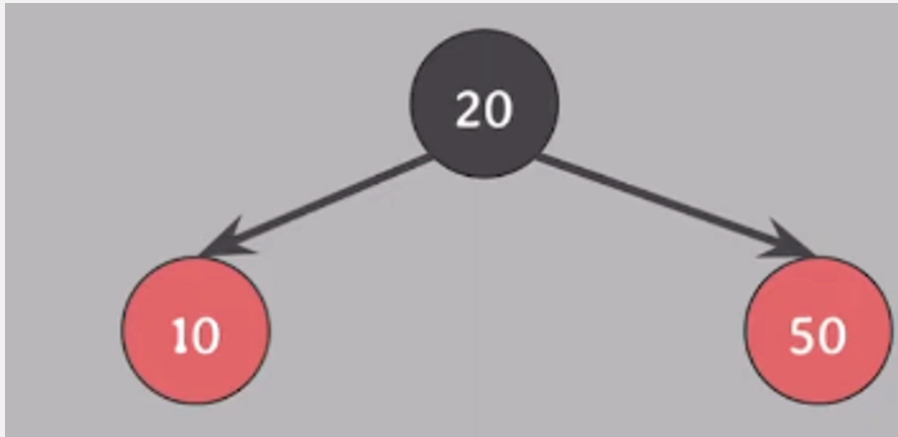
- 1. 꺾여 있는 부분을 펴 주기 위해서 20을 기준으로 왼쪽으로 회전한다.
- 2. 40과 50의 색을 바꾼다.
- 3. 50을 기준으로 오른쪽으로 회전한다.
- -> 4번 속성 포함 모든 속성을 만족함.

## CASE.2 해결하기

- Red 삽입 후 4번 속성을 위반했을 때
- 삽입된 red노드가 부모의 **오른쪽** 자녀
- &부모도 red고 할아버지의 **왼쪽** 자녀
- & 삼촌은 black이라면
- 부모를 기준으로 왼쪽으로 회전한 뒤 case.3의 방식으로 해결
- \* 왼,오 바뀌도 성립



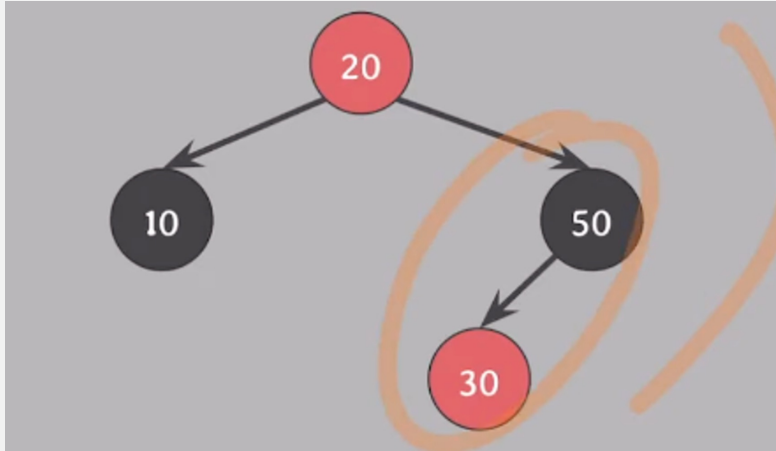
# INSERT



- RB트리 4번 속성 위반 red가 한쪽으로 몰려있지 않아서 옮길 수 없다. (삼촌도 빨간색)
- 4번 속성을 만족 시키면서 5번 속성을 유지하려면 10과 50을 black으로 바꾸고 20을 red로 바꾸면 된다.

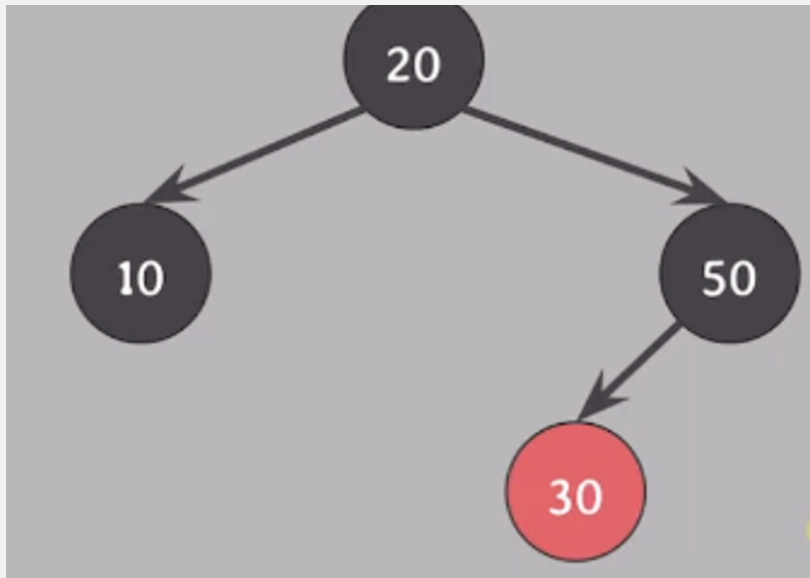


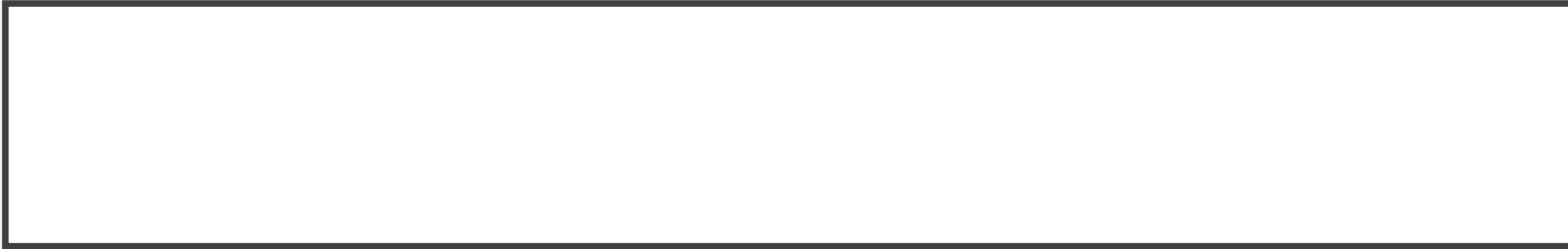
# INSERT



- → 4번,5번 속성 문제 해결
- 하지만 20이 루트노드라서 2번속성을 위반
- 20 을 black으로 바꿔주자

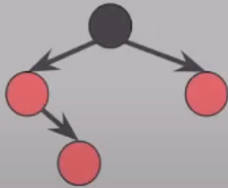
INSERT





red 삽입 후 #4 속성을 위반했을 때

### case.1 해결하기



#4 : 노드가 red라면 자녀들은 black

- 삽입된 red노드의 부모도red & 삼촌도red라면 부모와 삼촌을 black으로 바꾸고 할아버지를 red로 바꾼 뒤 할아버지에서 다시 확인을 시작한다.