레드 블랙 트리 (Red-Black Tree)

❖ 레드 블랙 트리도 이진탐색트리

- 이진탐색트리의 특징을 그대로 따름
- 레드 블랙 트리는 균형잡힌 이진탐색트리(balanced binary search tree)
- 탐색(search) 연산의 시간 복잡도 $O(\log n)$

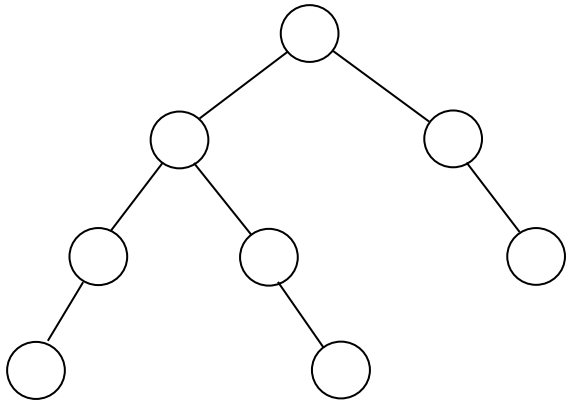
❖ 이진검색트리의 모든 노드에 블랙 또는 레드의 색을 칠하되 다음의 레드 블랙 특성을 만족해야 한다.

① *루트는 블랙이다.*
② *모든 리프는 블랙이다.*
③ *노드가 레드이면 그 노드의 자식은 반드시 블랙이다. (빨간색 노드가 연속으로 나올 수 없다.)*
④ *루트 노드에서 임의의 리프 노드에 이르는 경로에서 만나는 블랙 노드의 수는 모두 같다.*

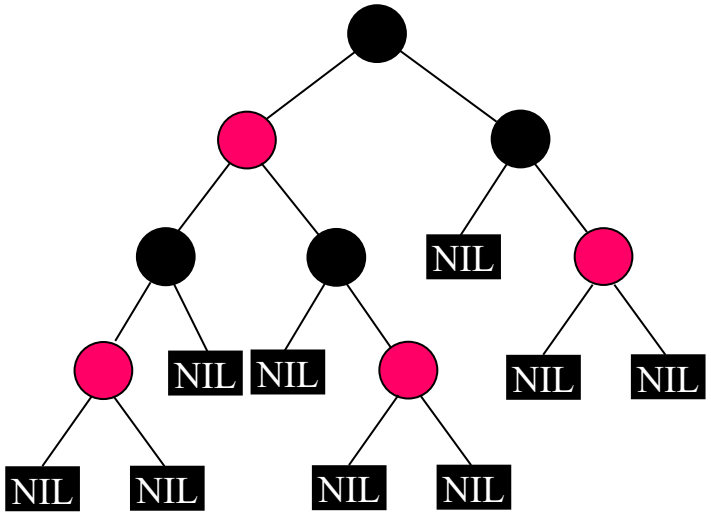
✓ 여기서 리프 노드는 일반적인 의미의 리프 노드와 다르다.
모든 NIL 포인터가 NIL이라는 리프 노드를 가리킨다고 가정한다.

레드 블랙 트리 (Red-Black Tree)

❖ BST vs RBT



(a) 이진검색트리의 한 예



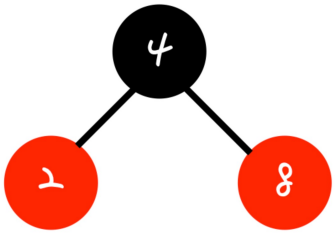
(b) (a)를 레드블랙트리로 만든 예

레드 블랙 트리 (Red-Black Tree)

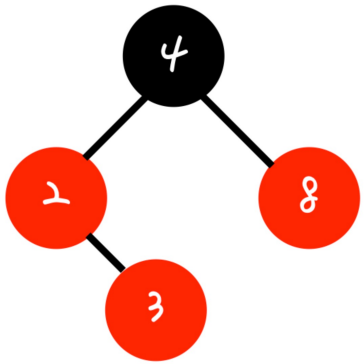
❖ 노드 삽입

- 첫 번째 (Root)노드 4 삽입
  - 블랙 적용 ⇒ 1번 조건 적용

- 다음 노드 2, 8 삽입
  - 레드 적용 ⇒ 3번 조건 위반 가능



- 노드 3을 레드로 추가
  - 3인 왜 노드 2의 오른쪽 자식노드에 위채하는가?
  - 노드 3을 레드로 했을 때 무엇이 문제인가?



레드 블랙 트리 (Red-Black Tree)

❖ 이중 레드 (Double Red) 문제 해결

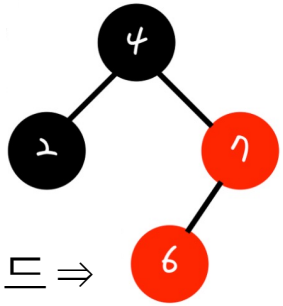
- Restructuring
  - 삽입된 노드, 부모 노드, 부모의 부모 노드를 오름차순으로 정렬
  - 중앙 값을 부모 노드로 만들고 나머지 노드를 자식으로 변환
  - 부모 노드가 된 노드를 검정으로, 나머지 노드를 빨강으로 변환

- Recoloring
  - 삽입된 노드의 부모와 삼촌 노드를 검정으로, 부모의 부모 노드를 빨강으로 Coloring
  - 부모의 부모 노드가 루트 (Root) 노드라면 그대로 검정으로 둠
  - 만약 이중 레드가 다시 발생하는 경우 다시 규칙에 맞게 Recoloring 혹은 Restructuring을 수행

레드 블랙 트리 (Red-Black Tree)

❖ 이중 레드 해결 방법

- 현재 삽입한 노드에 대한 부모의 형제 노드의 색깔에 따라 수행 방법을 달리함

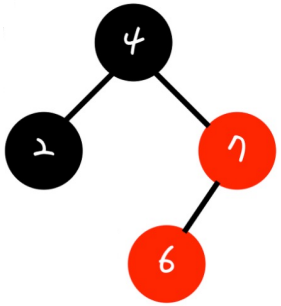


부모 노드: 7

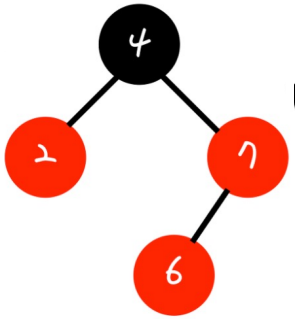
부모의 형제 노드: 2

입력 삽입한 노드⇒

- 부모의 형제 노드가 블랙인 경우 ⇒ Restructuring

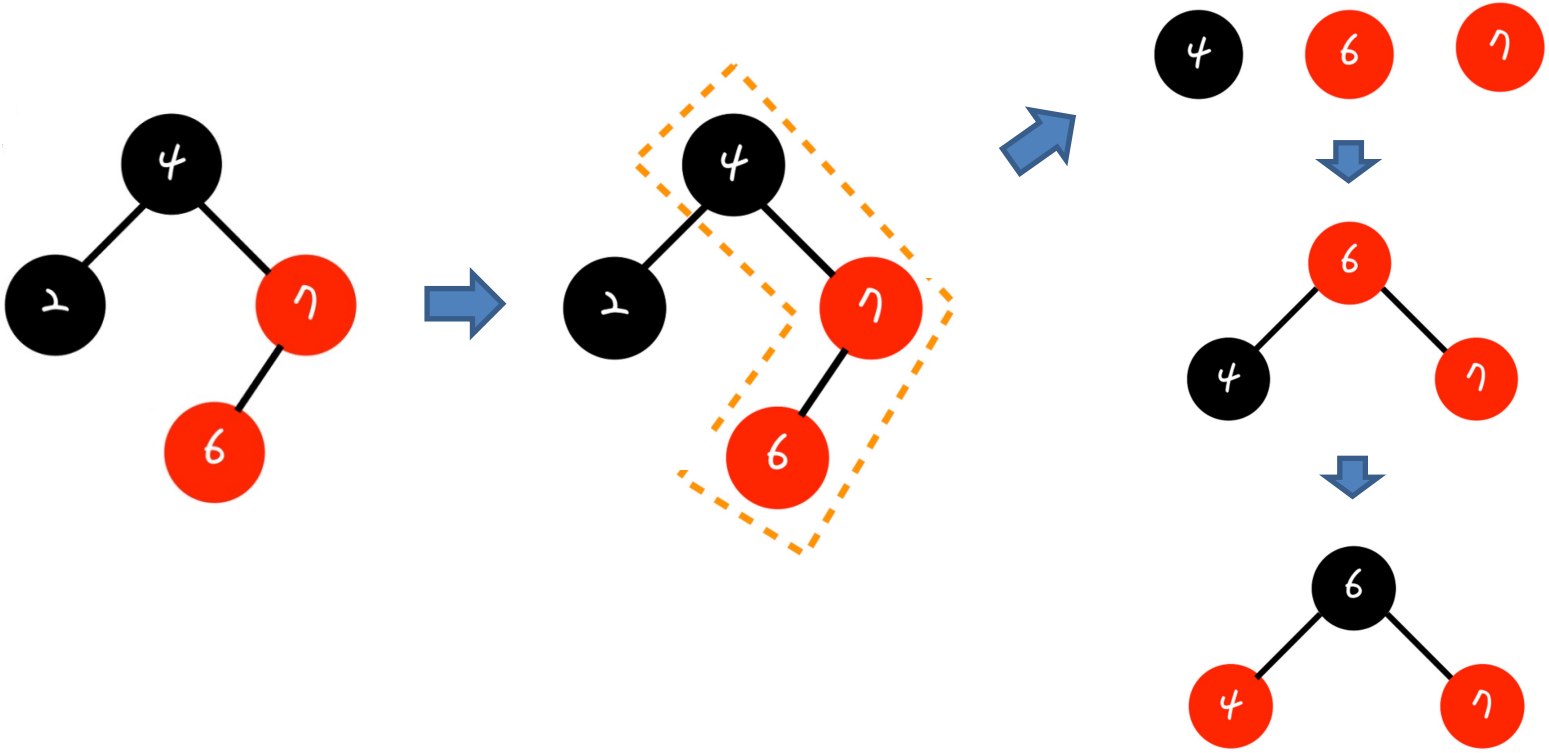


- 부모의 형제 노드가 레드인 경우 ⇒ Recoloring



레드 블랙 트리 (Red-Black Tree)

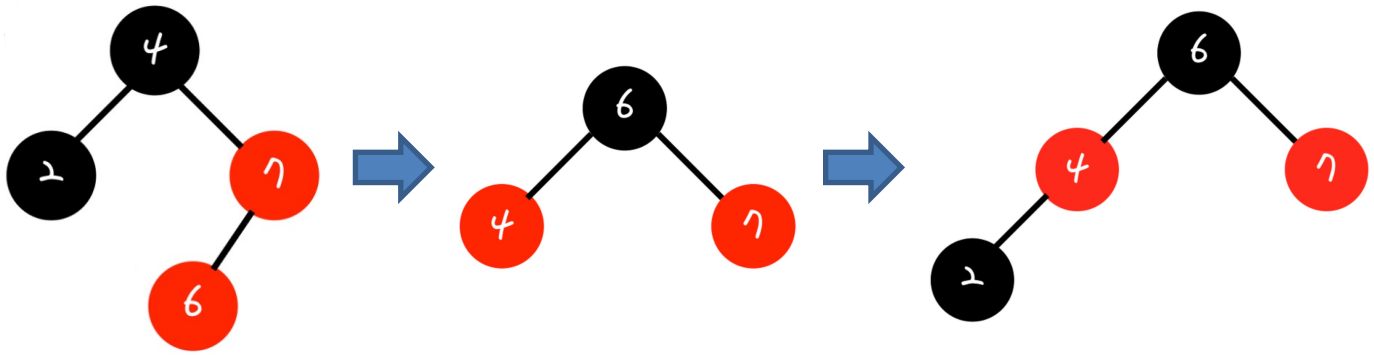
- **부모의 형제 노드가 블랙인 경우 ⇒ Restructuring**
  - 현재 노드와 부모 노드, 부모의 부모 노드를 오름차순으로 정렬
  - 오름차순 정렬 결과에서 가운데 있는 노드를 부모 노드, 나머지 노드를 자식 노드로 지정
  - 가운데 있는 노드는 블랙, 두 자식 노드는 레드로 지정



레드 블랙 트리 (Red-Black Tree)

❖ 부모의 형제 노드가 블랙인 경우 ⇒ Restructuring

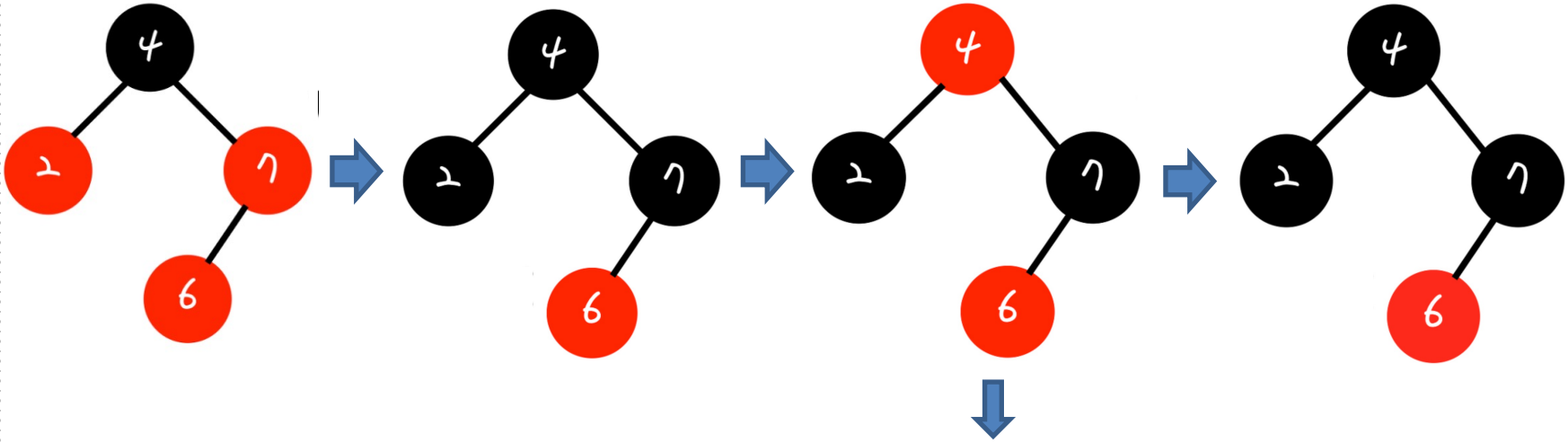
- 기존 노드 4의 자식 노드 2를 노드 4에 연결



- 다른 서브트리에 영향을 끼치지 않기 때문에 한 번의 Restructuring이면 완료
- 이중 레드를 해결하기 전과 후의 블랙 노드의 개수에 변화가 없다.
- Restructuring 자체의 시간복잡도는 $O(1)$ ⇒ 노드 순서 결정, 트리 만드는 시간, 원래 노드의 구조 변환 시간 모두 상수 시간
- 삽입할 노드가 들어갈 위치를 찾아 해당 노드를 삽입한 뒤 Restructuring이 일어나므로 총 수행시간은 $\boldsymbol{O(\log n)}$

레드 블랙 트리 (Red-Black Tree)

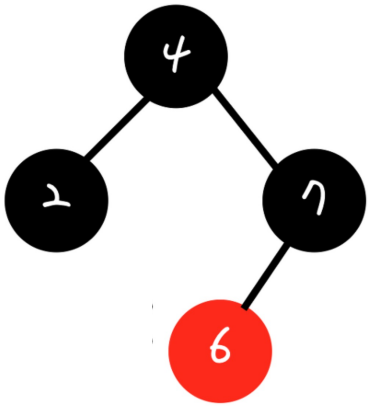
- **부모의 형제 노드가 레드인 경우 ⇒ Recoloring**
  - 현재 삽입된 노드의 부모 노드와 부모의 형제 노드를 블랙으로 지정
  - 부모의 부모 노드를 레드로 지정
  - 부모의 부모 노드가 루트 노드가 아닐 경우 이중 레드 문제가 다시 발생될 수 있음
  - Recoloring은 한 번에 안 끝날 수 있음



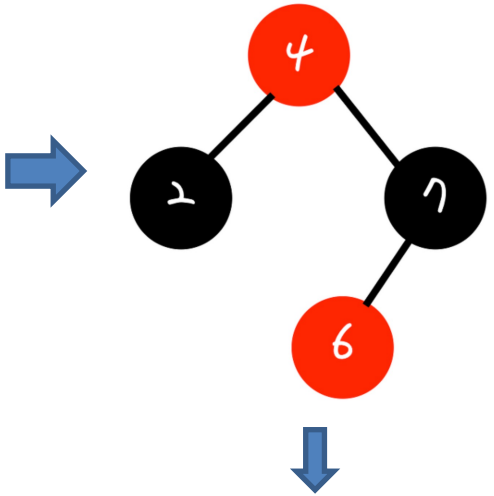
부모의 부모 노드인 4가 루트 노드였다면 블랙이어야 함 ⇒ 1번 조건

레드 블랙 트리 (Red-Black Tree)

❖ 부모의 형제 노드가 레드인 경우 ⇒ Recoloring



블랙 노드 4가 루트 노드가 아니라 다른 어떤 트리의 서브 트리였다면 블랙 노드 4는 레드 노드 4가 됨



- 레드 노드 4가 루트 노드가 아니므로 레드 노드 4의 또 다른 부모 노드가 있을 수 있고, 이 노드가 레드라면 이중 레드의 문제가 다시 발생
- 최악의 경우 루트 노드까지 올라가면서 Recoloring 수행할 수 있음

- 레드 노드 4의 부모 노드의 형제 노드 색깔이 블랙이면 Restructuring, 레드이면 Recoloring을 수행하면서 더 이상 이중 레드 문제가 발생되지 않을 때까지 반복

레드 블랙 트리 (Red-Black Tree)

❖ Recoloring의 시간복잡도

- Recoloring 자체의 시간복잡도는 $O(1)$
- Restructuring과 마찬가지로 새로운 노드를 삽입하고자 할 때 삽입할 위치를 먼저 찾아야 하므로 $O(\log n)$ 소요

레드 블랙 트리 (Red-Black Tree)

- **레브 블랙 트리는 균형 잡힌 이진 트리인가?**
  - 조건 4: 루트 노드에서 임의의 리프 노드에 이르는 경로에서 만나는 블랙 노드의 수는 모두 같다. ⇔ 모든 리프노드에서 블랙 깊이는 같다.

가장 짧은 블랙 깊이

가장 긴 블랙 깊이

일반적으로 기껏해야 2배 차이가 남