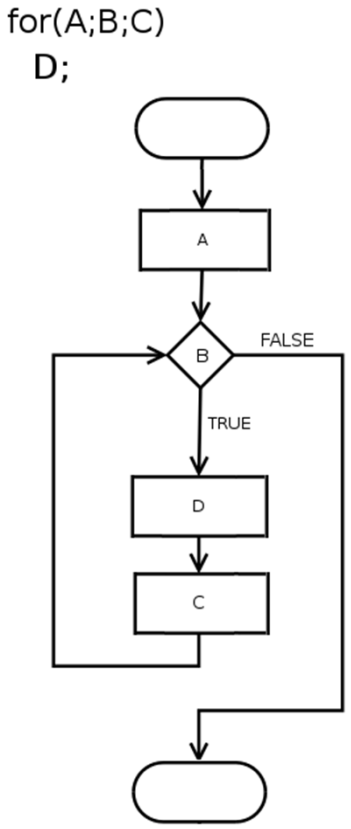
Clickseo.com

자료구조 및 알고리즘

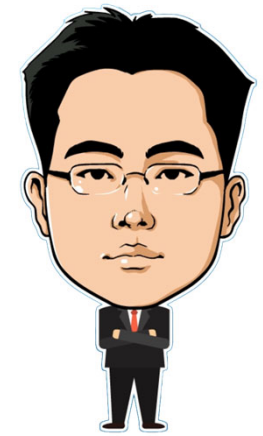


검색 트리
(Search Tree)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



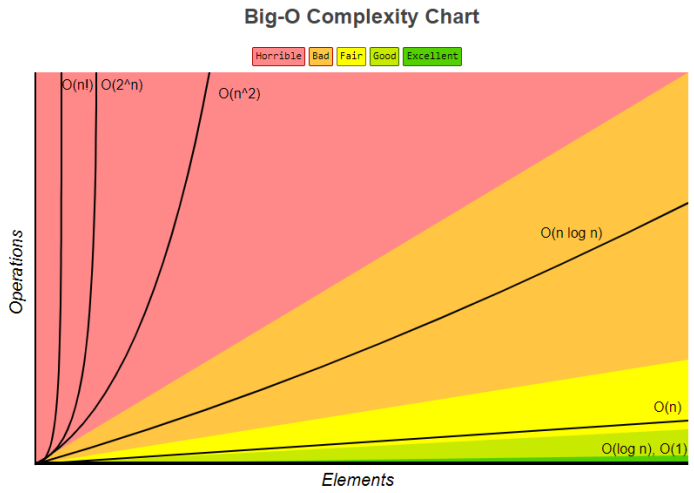
목 차

백문이불여일타(百聞而不如一打)

- 이진 검색 트리
- 균형 이진 검색 트리
- 균형 다진 검색 트리

검색 트리 (1/2)

- **자료 구조:** 시간 복잡도
  - 배열 또는 연결 리스트: O(n), 평균 Θ(n)
  - **이진 검색 트리:** 검색, 삽입, 삭제 시 **평균 Θ(logN), 최악의 경우 O(N)**
  - **균형 이진 검색 트리**
    - 검색, 삽입, 삭제 시 **최악의 경우 O(logN)**
    - AVL 트리, RB 트리
  - **균형 다진 검색 트리**
    - 검색, 삽입, 삭제 시 **최악의 경우 O(logN)**
    - 2-3 트리, 2-3-4 트리, B-트리
  - 해시 테이블
    - 검색, 삽입, 삭제 시 평균 O(1)



검색 트리 (2/2)

- **검색 트리**(Binary Search Tree)
  - 검색 트리: 이진 검색 트리, 다진 검색 트리

[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

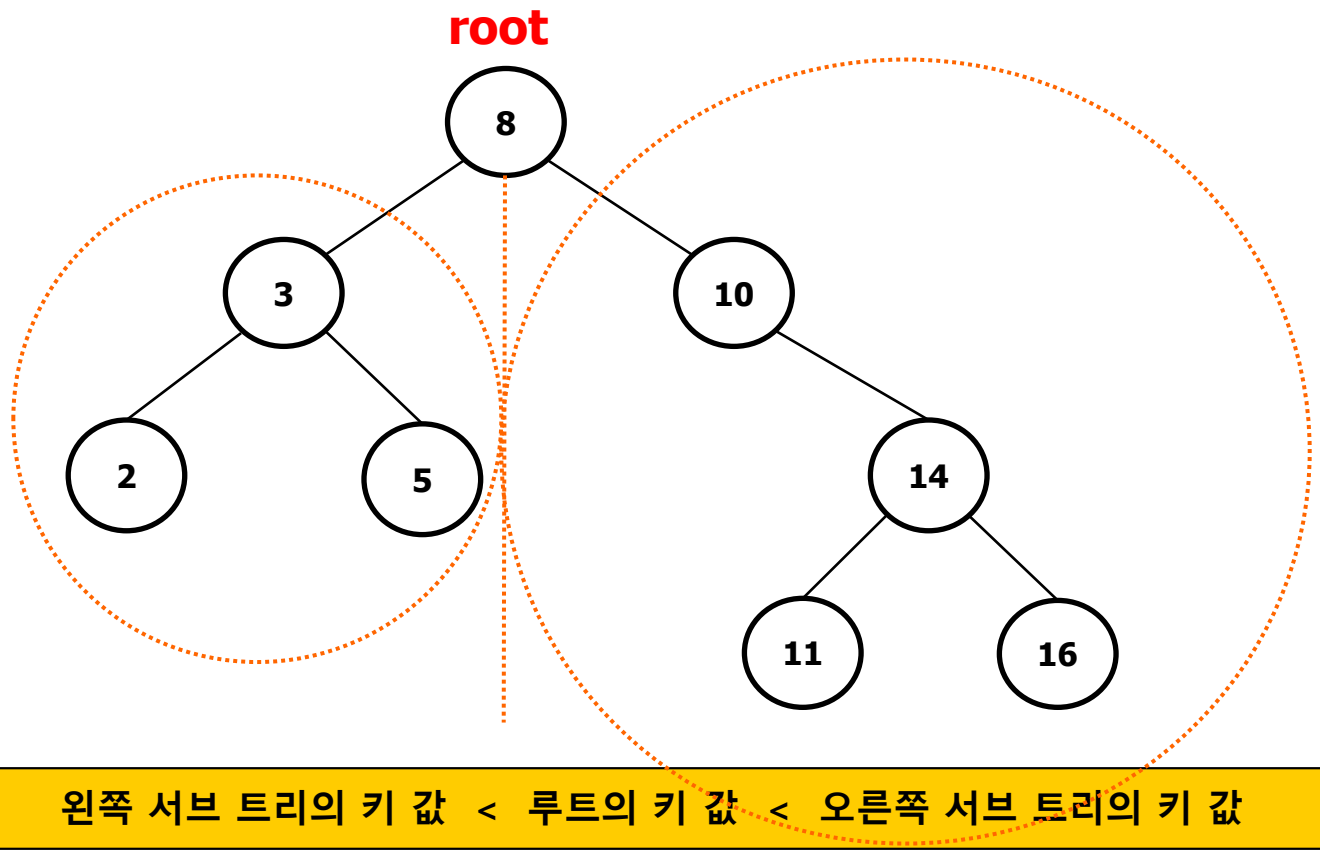
이진 검색 트리

백문이불여일타(百聞而不如一打)

- 이진 검색 트리
  - 이진 검색 트리 연산
- 균형 이진 검색 트리
- 균형 다진 검색 트리

이진 검색 트리 (1/3)

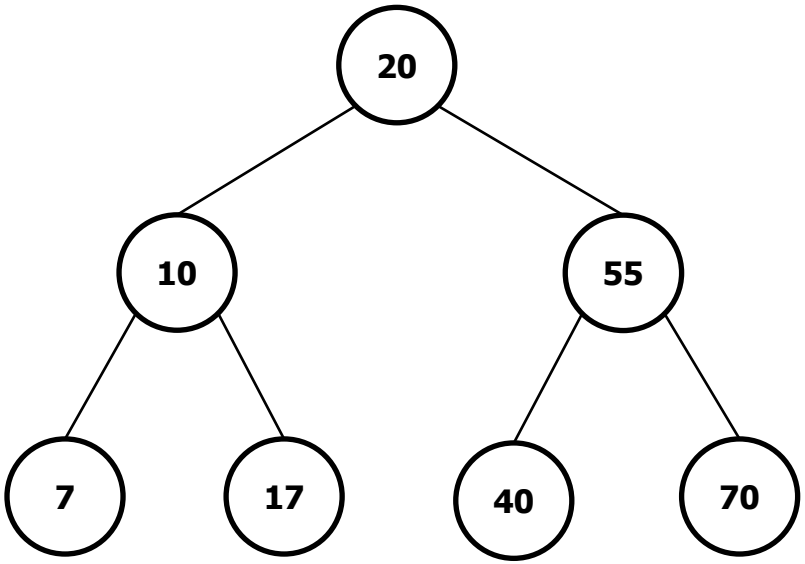
- **이진 검색 트리**(Binary Search Tree)
  - 모든 노드는 서로 다른 키를 갖는다(유일한 키 값).
    - 각 노드는 최대 2개의 자식을 갖는다.

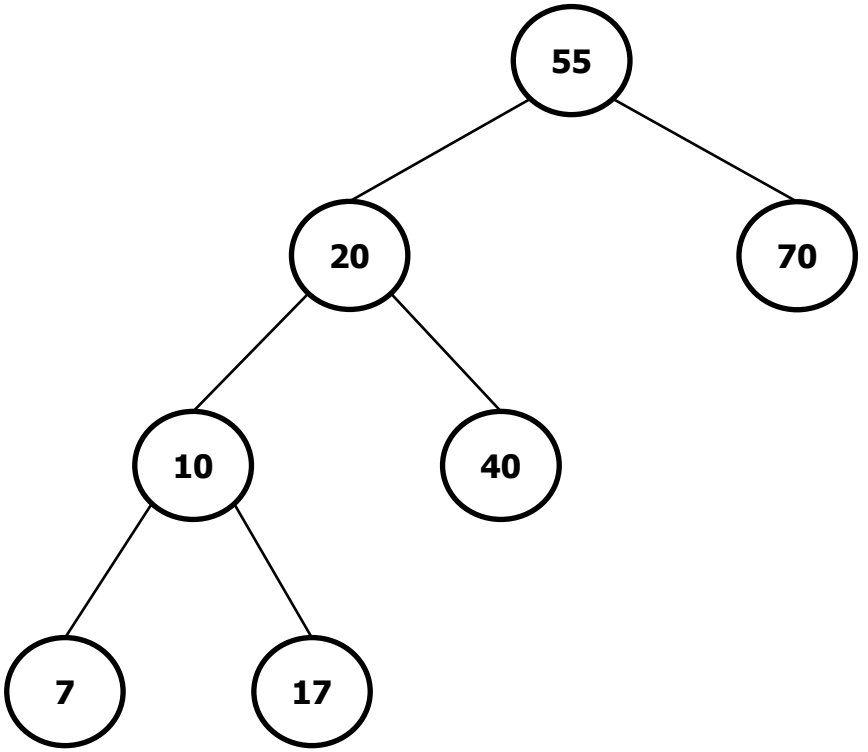


왼쪽 서브 트리의 키 값 < 루트의 키 값 < 오른쪽 서브 트리의 키 값

이진 검색 트리 (2/3)

- 이진 검색 트리
  - 같은 데이터와 다른 이진 검색 트리 #1

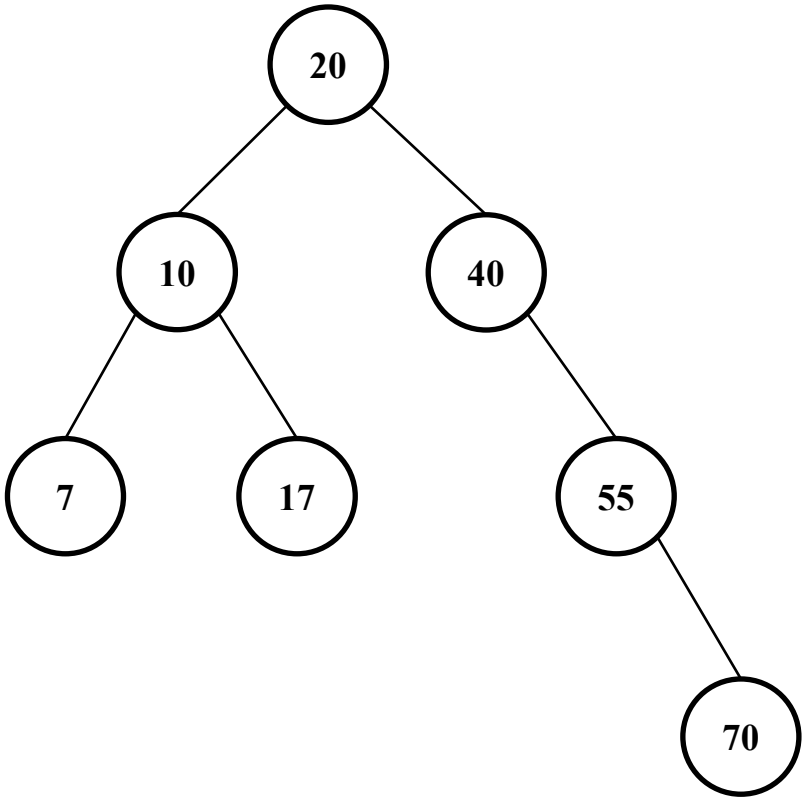


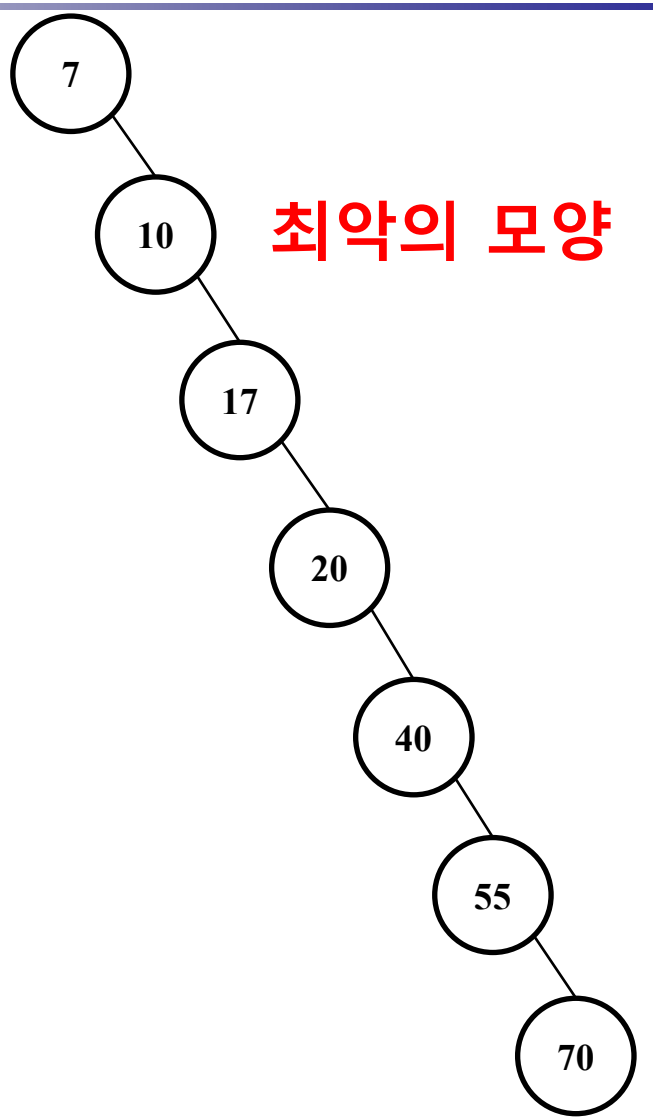


최선의 모양

이진 검색 트리 (3/3)

- 이진 검색 트리
  - 같은 데이터와 다른 이진 검색 트리 #2





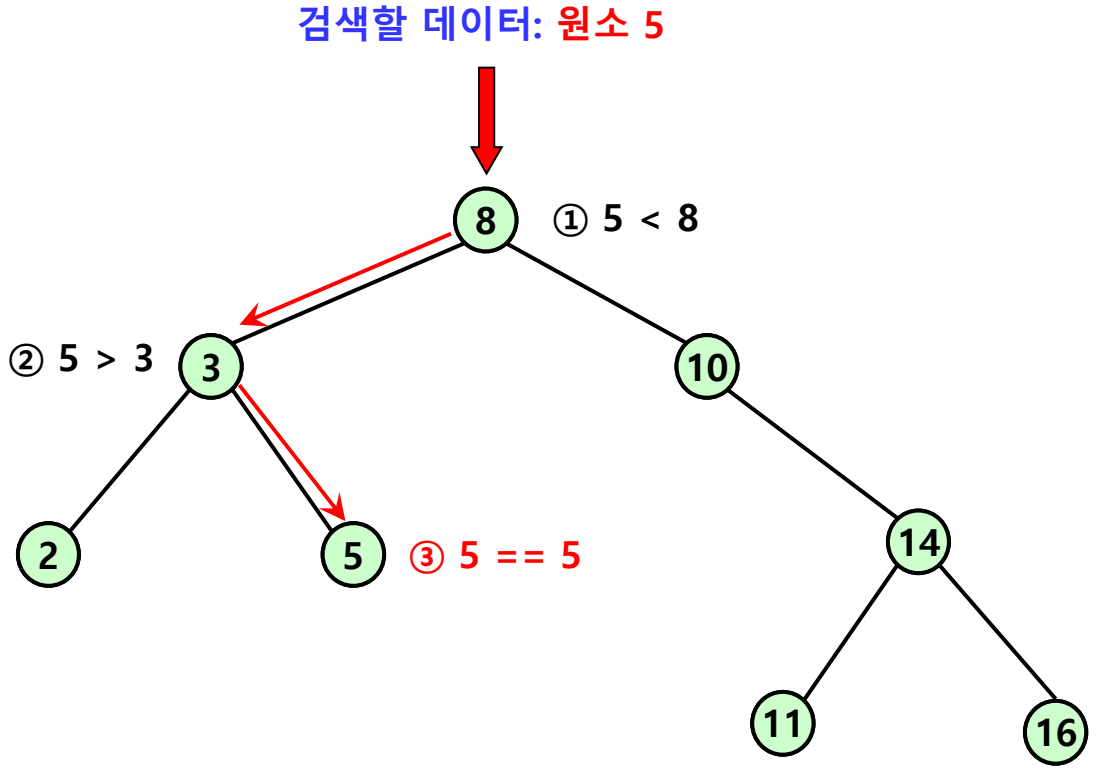
최악의 모양

이진 검색 트리

이진 검색 트리 연산: 검색, 삽입, 삭제

이진 검색 트리 (1/7)

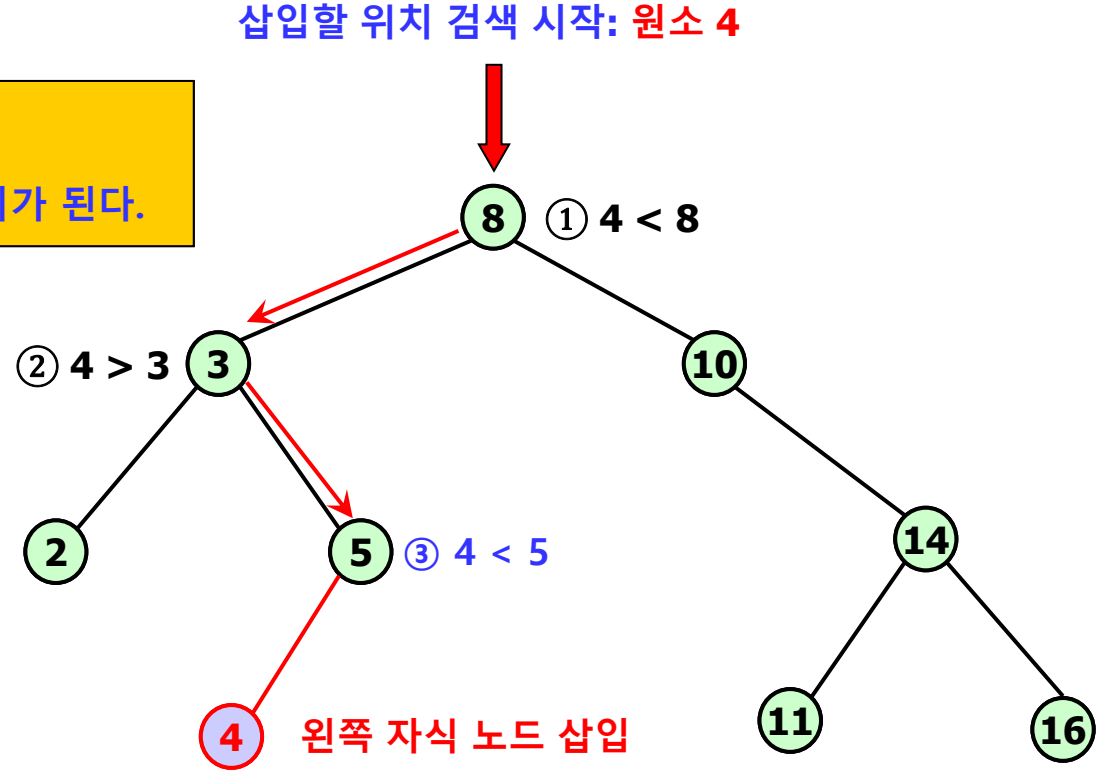
- **이진 검색 트리: 탐색**
  - 검색 과정



이진 검색 트리 (2/7)

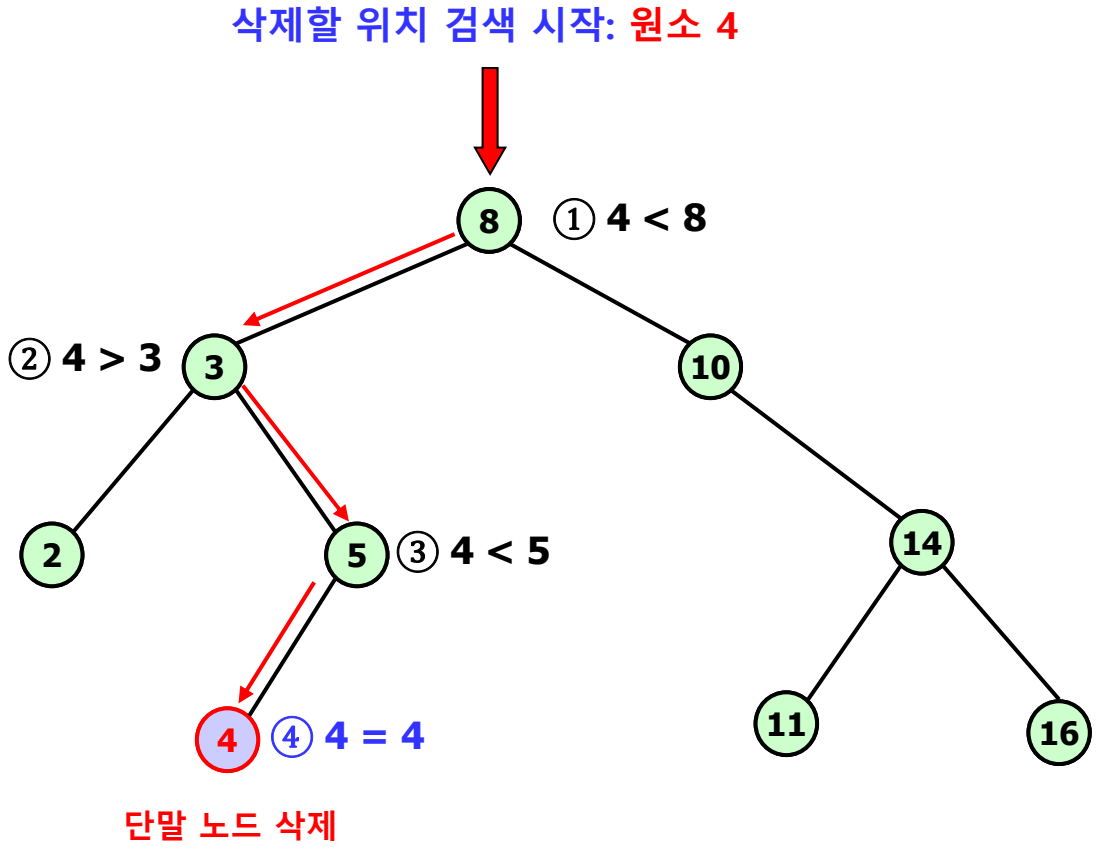
- **이진 검색 트리: 삽입**
  - 삽입 과정
    1. 삽입할 노드의 위치(부모 노드의 주소) 검색
    2. 노드 삽입

"검색 실패가 결정 된 위치 "
즉, 왼쪽 자식 노드의 위치가 삽입 할 자리가 된다.



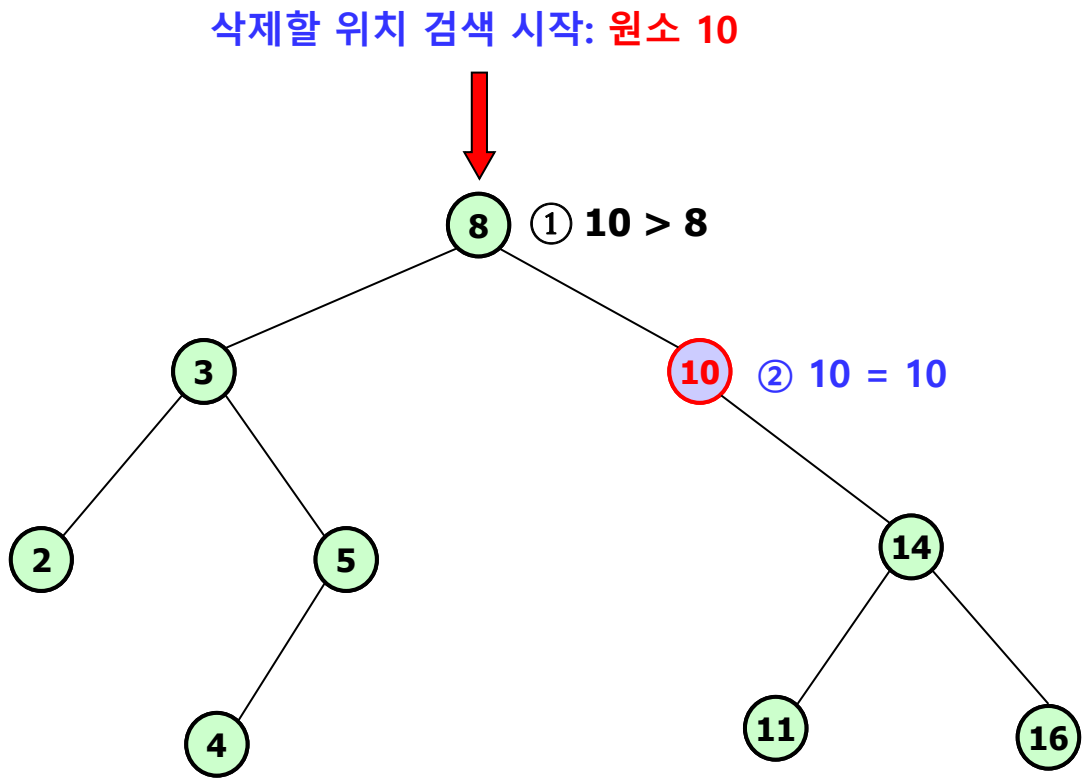
이진 검색 트리 (3/7)

- **이진 검색 트리: 삭제 #1**
  - 삭제 과정: 단말 노드



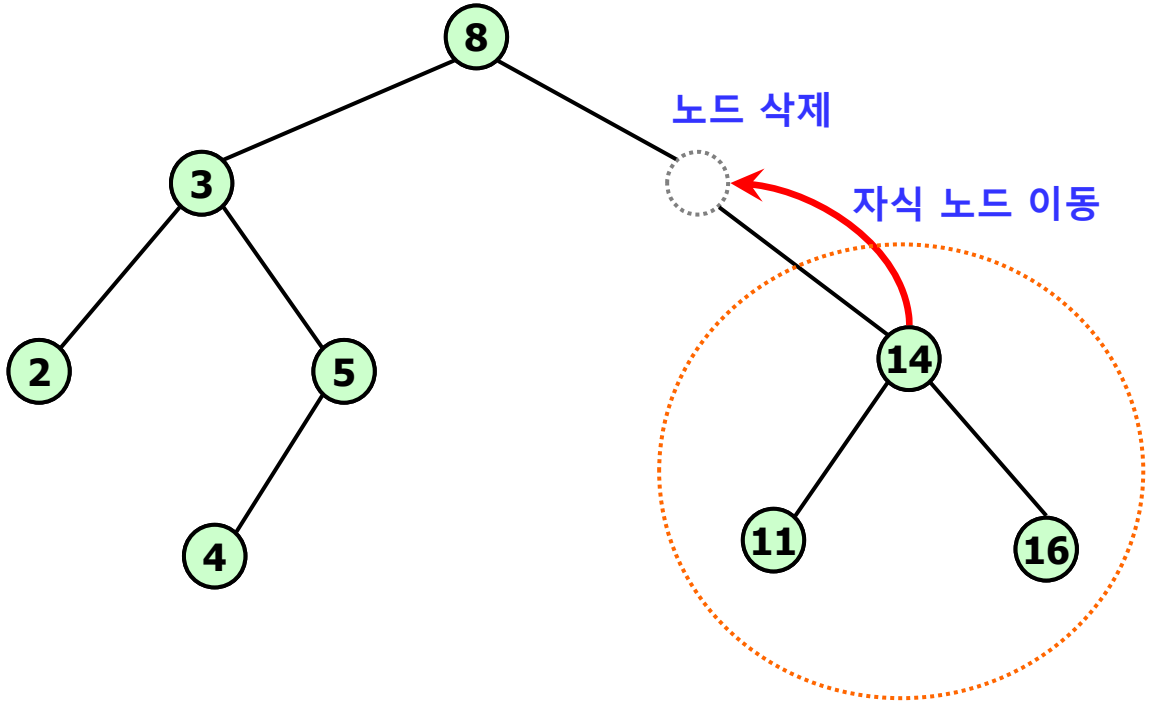
이진 검색 트리 (4/7)

- **이진 검색 트리: 삭제 #2**
  - 삭제 과정: 하나의 자식 노드만 존재
    1. 삭제할 노드의 검색



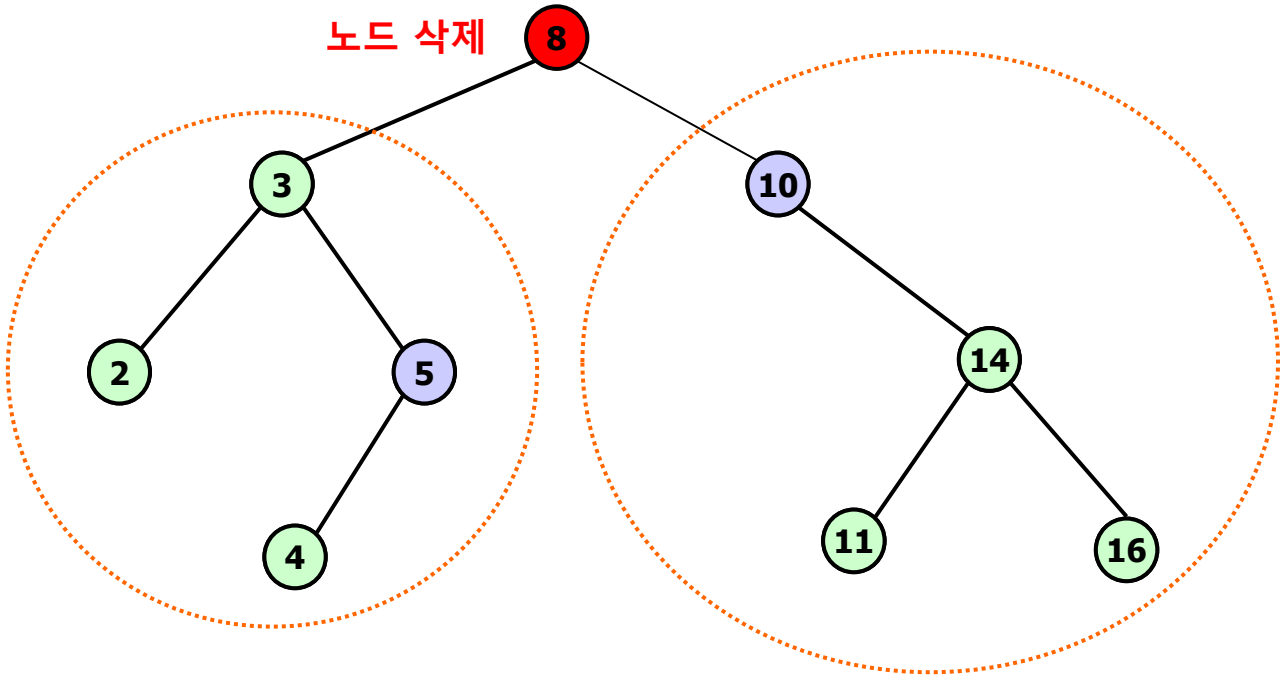
이진 검색 트리 (5/7)

- **이진 검색 트리: 삭제 #2**
  - 삭제 과정: 하나의 자식 노드만 존재
    2. 삭제할 노드의 삭제 및 위치 조정



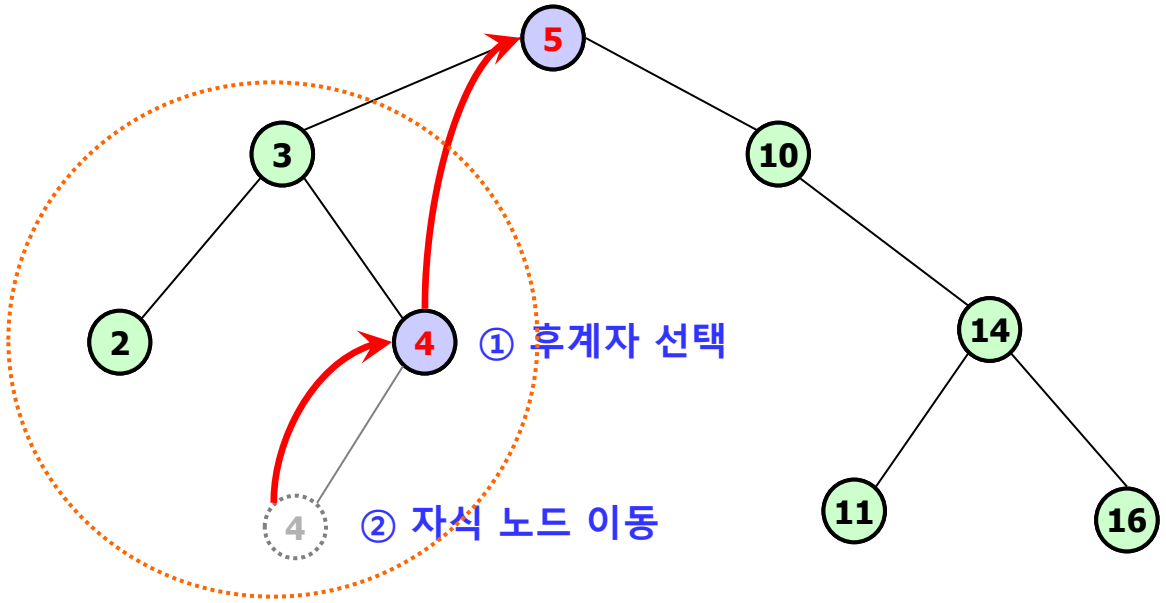
이진 검색 트리 (6/7)

- **이진 검색 트리: 삭제 #3**
  - 삭제 과정: 두 개의 자식 노드가 존재
    1. 삭제할 노드의 탐색 및 후계자 노드 선정
       - 왼쪽 서브 트리에서 가장 큰 키 값을 가진 노드
       - 오른쪽 서브 트리에서 가장 작은 키 값을 가진 노드



이진 검색 트리 (7/7)

- **이진 검색 트리: 삭제 #3**
  - 삭제 과정: 두 개의 자식 노드가 존재
    2. **트리 재구성:** 데이터 5를 가진 노드를 후계자로 선택한 경우



이진 검색 트리

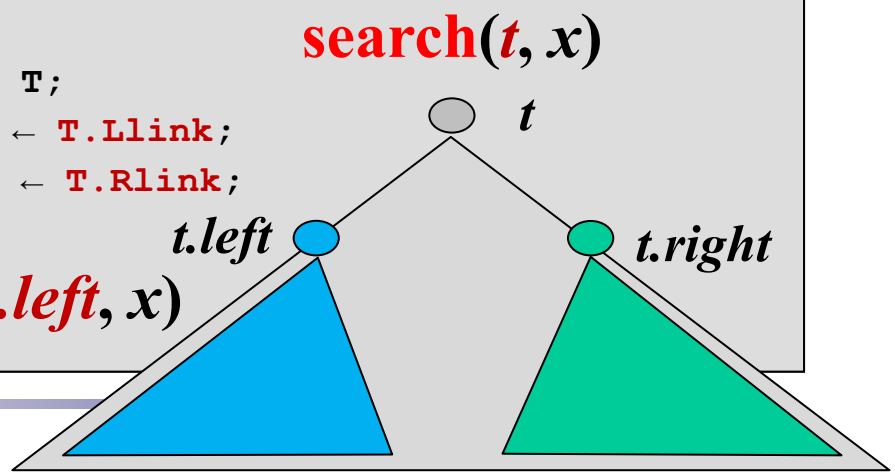
이진 검색 트리 연산: 알고리즘

이진 검색 트리 연산: 알고리즘 (1/3)

- 이진 검색 트리 연산: 알고리즘(검색)

```
// 재귀적 용법
searchBST(T, data)
    if (T = NULL or data = T.key)     then     return T;
    else if (data < T.key) then       return searchBST(T.Llink, data);
    else if (data > T.key) then       return searchBST(T.Rlink, data);
end searchBST()
```

```
// 비재귀적 용법
searchBST(T, data)
    while (T ≠ NULL) do
    {
          if (data = T.key) then return T;
          else if (data < T.key) then T ← T.Llink;
          else if (data > T.key) then T ← T.Rlink;
    }
    return NULL;
end searchBST()
```

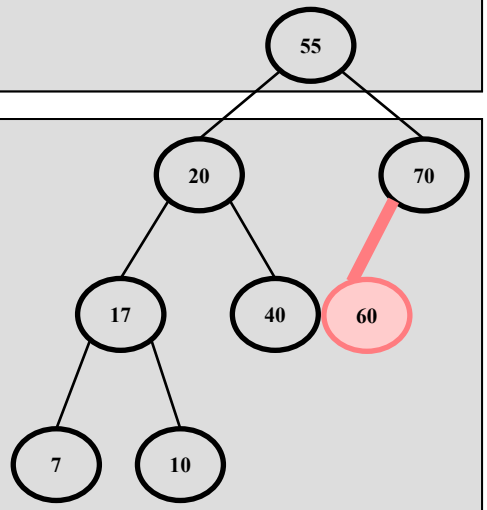


이진 검색 트리 연산: 알고리즘 (2/3)

- 이진 검색 트리 연산: 알고리즘(삽입)

```
// 재귀적 용법
insertBST(T, data)
    if (T = NULL) then T ← newDNode;
    else if (data < T.key) then      T.Llink = insertBST(T.Llink, data);
    else if (data > T.key) then      T.Rlink = insertBST(T.Rlink, data);
    return T;
end insertBST()
```

```
// 비재귀적 용법
insertBST(T, data)
    while (T ≠ NULL) do
    {
          if (data = T.key) then
                return error;
          parent ← T;
          if (data < T.key) then      T ← T.Llink;
          else                        T ← T.Rlink;
    }
    newDNode ← makeDNode(data);
    if (parent = NULL) then           T ← newDNode;
    else if (data < parent.key) then  parent.Llink ← newDNode;
    else if (data > parent.key) then  parent.Rlink ← newDNode;
end insertBST()
```



이진 검색 트리 연산: 알고리즘 (3/3)

- **이진 검색 트리 연산: 알고리즘(삭제)**

```
deleteBST(T, data)
    del ← 삭제할 노드;
    parent ← 삭제할 노드의 부모 노드;
    if (del = NULL) then return;
    if (del.Llink = NULL and del.Rlink = NULL) then {            // 단말 노드
           if (parent.Llink = del) then parent.Llink ← NULL;
           else parent.Rlink ← NULL;
    }
    else if (del.Llink = NULL or del.Rlink = NULL) then {        // 하나의 자식 노드
           if (del.Llink ≠ NULL) then {
                      if (parent.Llink = del) then parent.Llink ← del.Llink;
                      else parent.Rlink ← del.Llink;
           }
           else {
                      if (parent.Llink = del) then parent.Llink ← del.Rlink;
                      else parent.Rlink ← del.Rlink;
           }
    }
    else if (del.Llink ≠ NULL and del.Rlink ≠ NULL) {            // 두 개의 자식 노드
           max ← maxNode(del.Llink);          // min ← minNode(del.Rlink);
           del.key ← max.key;                 // del.key ← min.key;
           deleteBST(del.Llink, del.key);     // deleteBST(del.Rlink, del.key);

    }
    end deleteBST()
```

이진 검색 트리

이진 검색 트리 구현: 연결 자료 구조

– C/C++, Python

균형 이진 검색 트리

백문이불여일타(百聞而不如一打)

- 이진 검색 트리
- **균형 이진 검색 트리**
  - AVL 트리
  - 레드-블랙 트리
- 균형 다진 검색 트리

균형 검색 트리

- **균형 검색 트리**(Binary Search Tree)
  - 이진 검색 트리: AVL 트리, 레드-블랙 트리

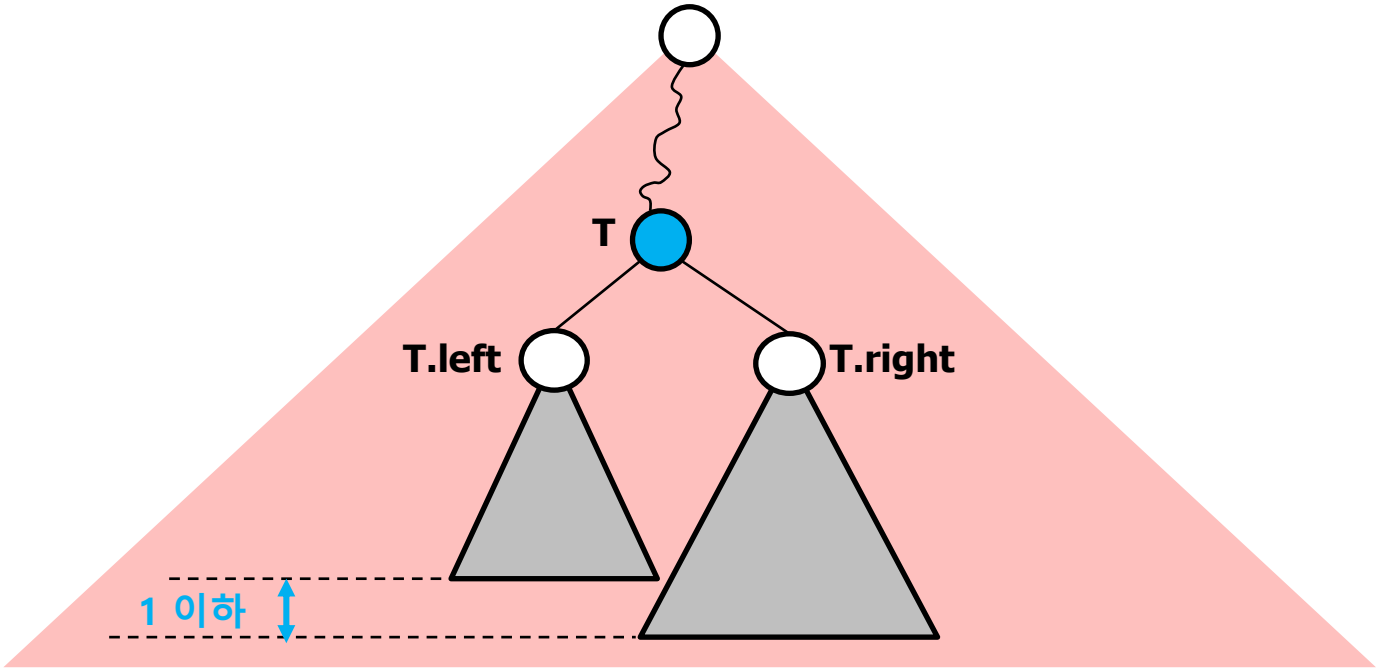
[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

균형 이진 검색 트리

AVL 트리

AVL 트리 (1/6)

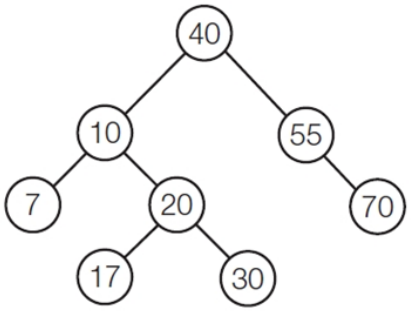
- **AVL 트리**
  - 전체 트리의 구조가 균형이 맞도록 하는 트리
    - 모든 노드에 대해 좌 서브 트리의 높이(깊이)와 우 서브 트리의 높이의 차가 1을 넘지 않는다(즉, 트리 구조가 한쪽으로 쏠리는 것을 막을 수 있다).



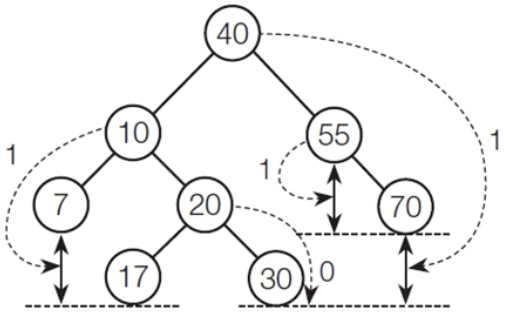
[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

AVL 트리 (2/6)

- **AVL 트리**
  - **AVL 트리의 예**

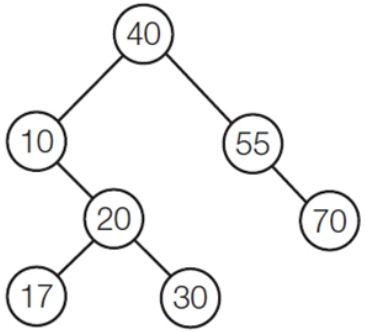


(a) AVL 트리의 예

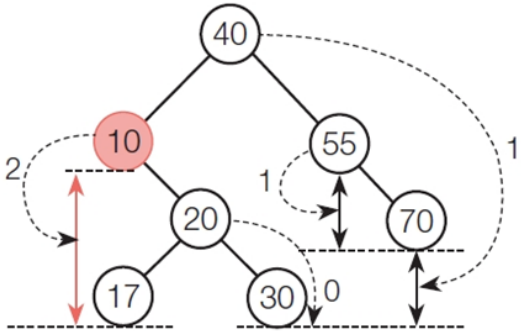


(b) 서브 트리의 높이 차

  - **AVL 트리가 아닌 예 #1**



(a) AVL 트리가 아닌 예

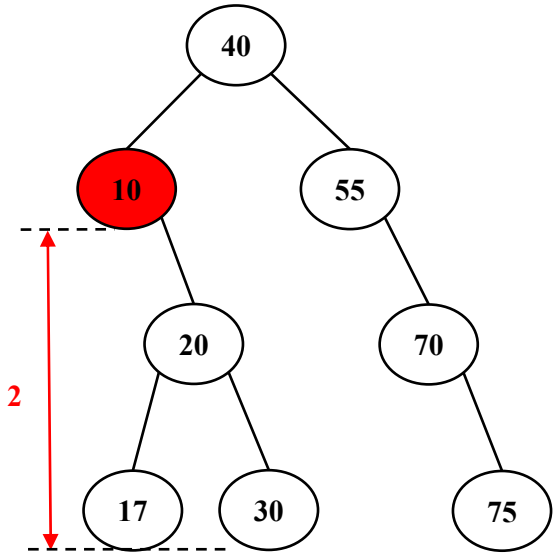


(b) AVL 트리가 아닌 이유: 높이 차 2

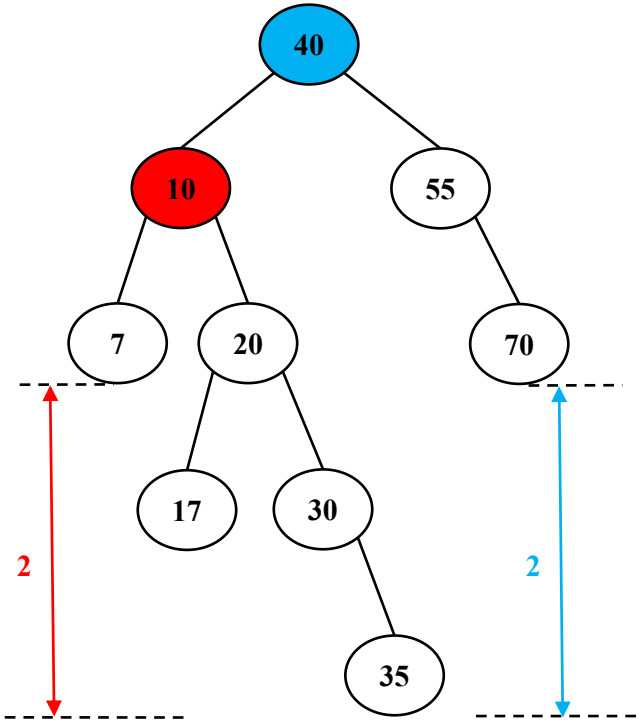
[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

AVL 트리 (3/6)

- **AVL 트리**
  - AVL 트리가 아닌 예 #2



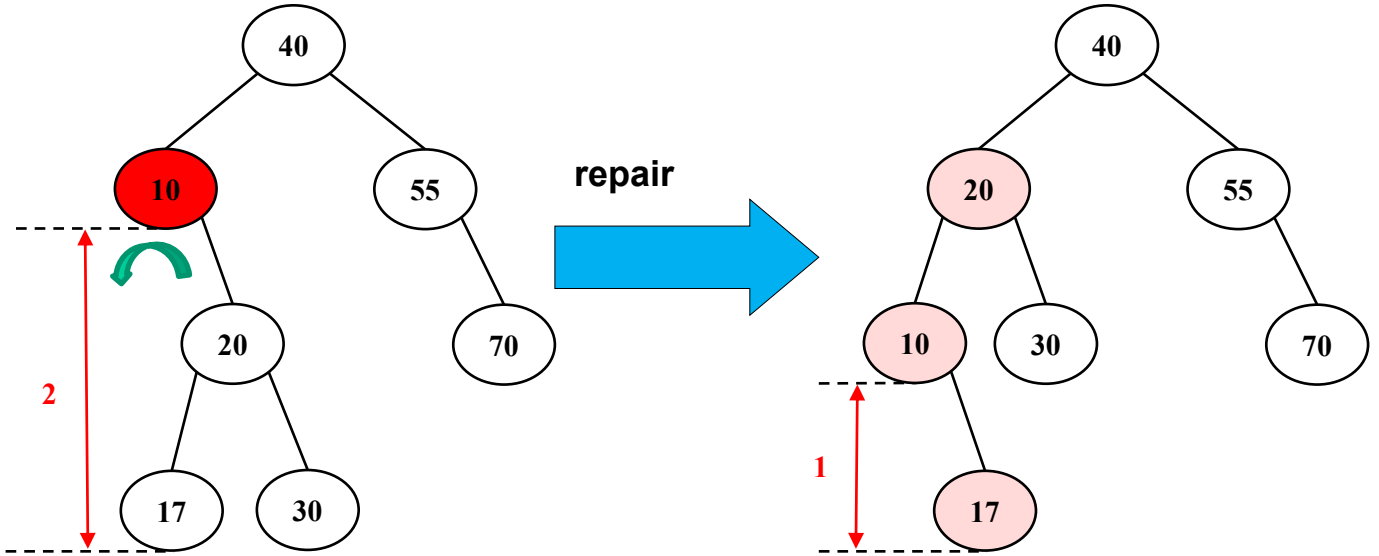
AVL Tree가 아니다



AVL Tree가 아니다

AVL 트리 (4/6)

- **AVL 트리: 균형 맞추기**
  - 균형 맞추기: 좌회전
    - 좌회전으로 불균형 해결

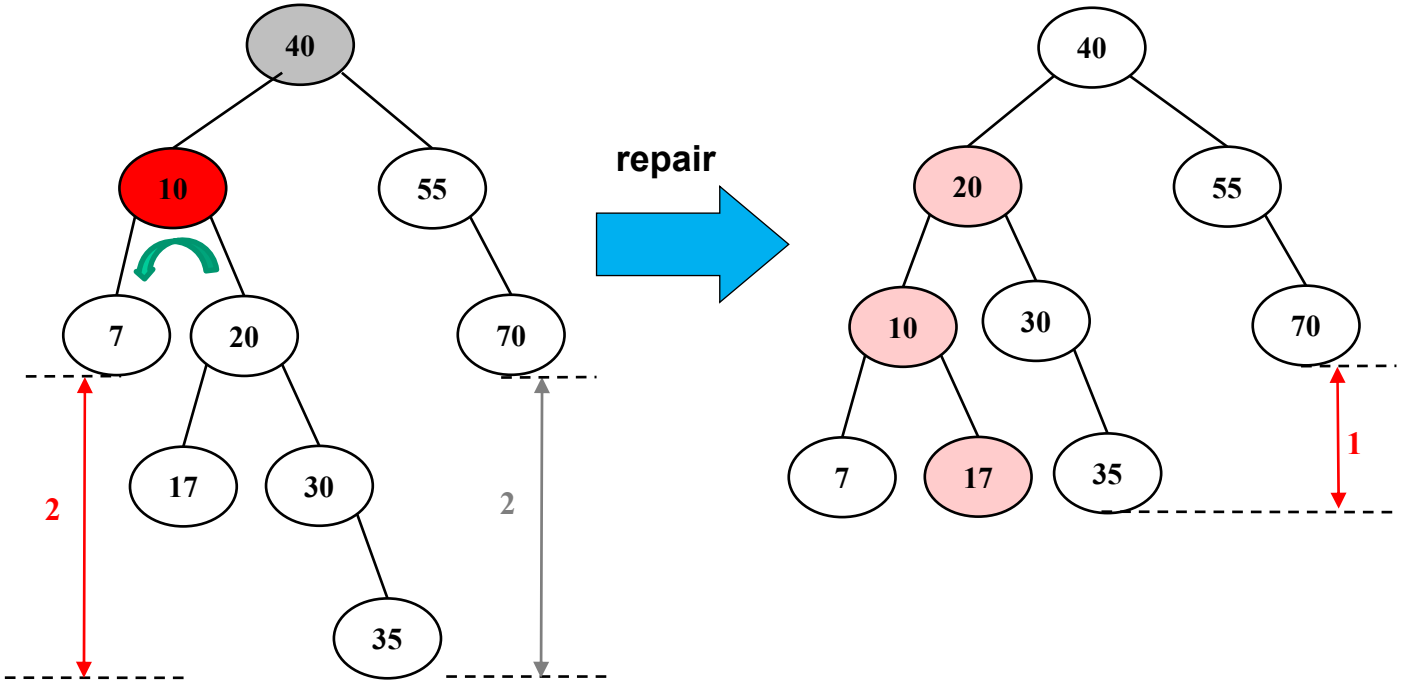


AVL Tree가 아닌 예

좌회전해서 AVL Tree로 수선됨

AVL 트리 (5/6)

- **AVL 트리: 균형 맞추기**
  - 균형 맞추기: 좌회전
    - 좌회전으로 두 곳의 불균형 해결

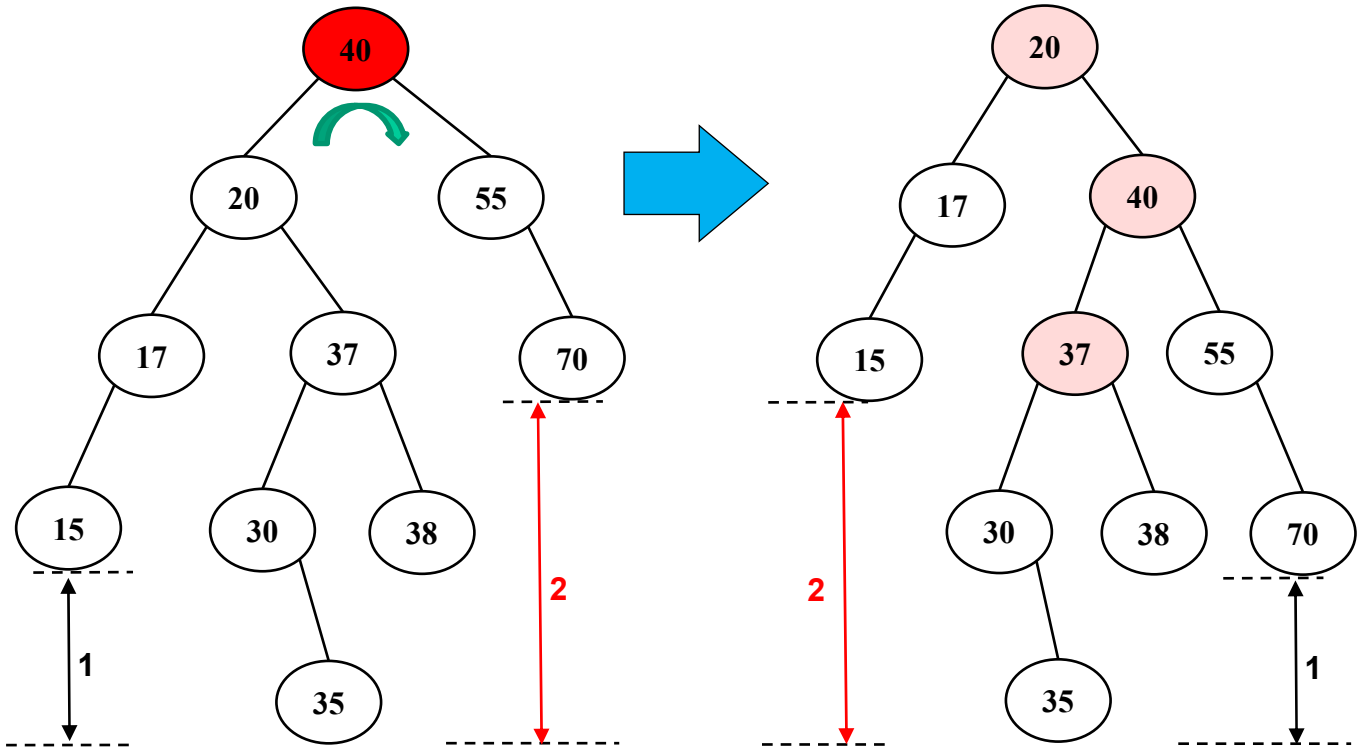


AVL Tree가 아닌 예

좌회전해서 AVL Tree로 수선됨

AVL 트리 (6/6)

- **AVL 트리:** 균형 맞추기
  - 균형 맞추기: 단순한 회전으로 해결이 안되는 경우



AVL Tree가 아닌 예

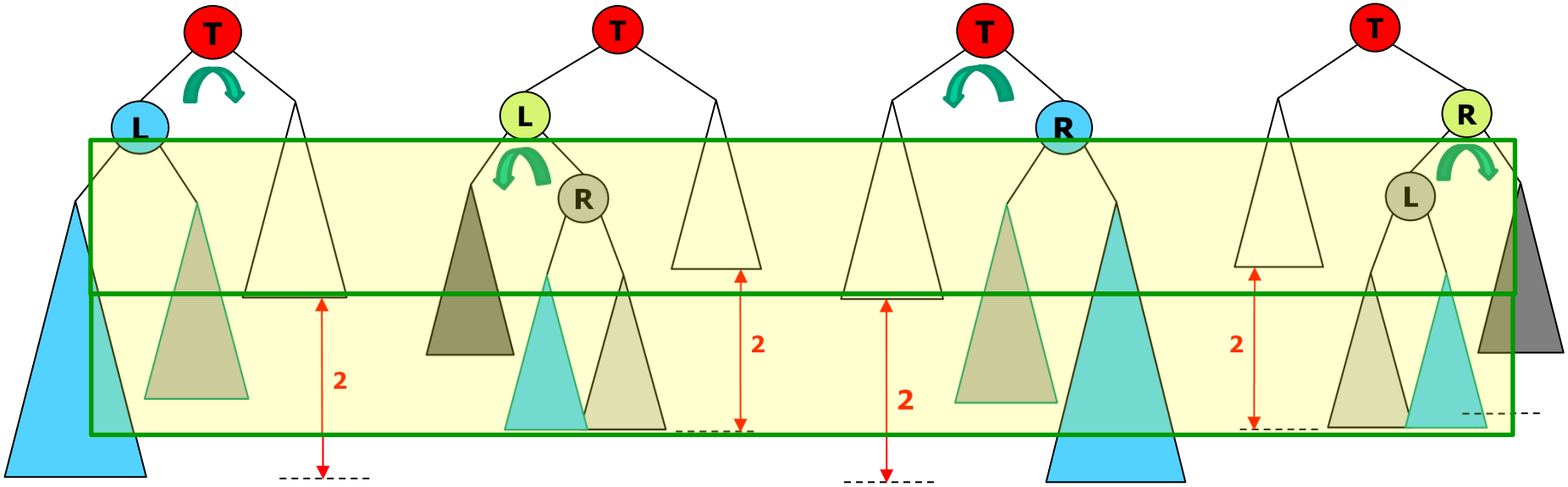
수선이 안된다

균형 이진 검색 트리

AVL 트리: 구성 및 구현

AVL 트리: 구성 및 구현 (1/11)

- **AVL 트리: 구성 및 구현**
  - 4가지 유형
    - LL : T.Llink.Llink 가 가장 깊음
    - LR : T.Llink.Rlink 가 가장 깊음
    - RR : T.Rlink.Rlink 가 가장 깊음
    - RL : T.Rlink.LLlink 가 가장 깊음



Type LL Type LR Type RR Type RL

AVL 트리: 구성 및 구현 (2/11)

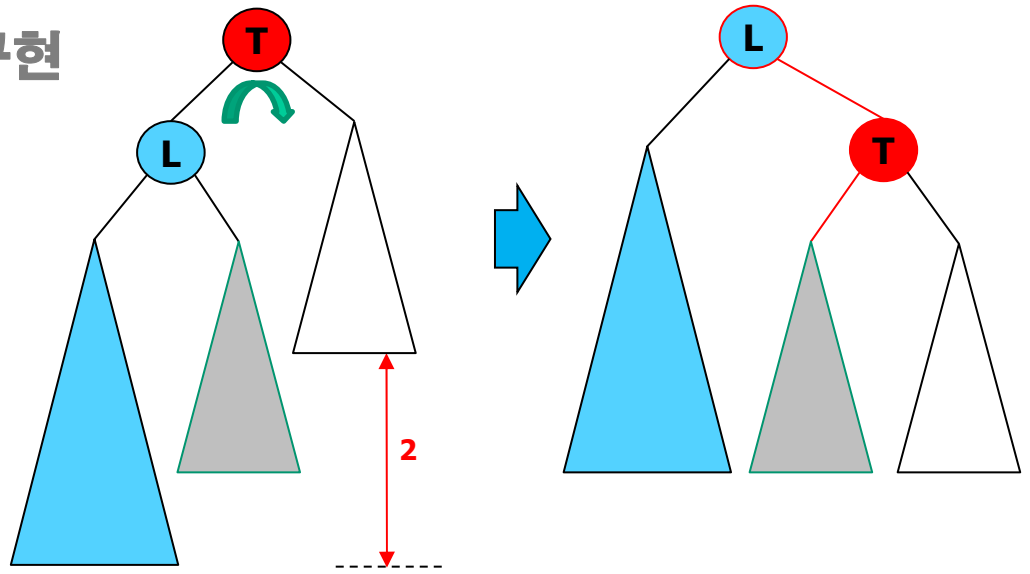
- **AVL 트리: 구성 및 구현**
  - 균형 맞추기
    - 좌회전(Left Rotation)
    - 우회전(Right Rotation)

```
// AVL 트리의 균형 잡기
balanceAVL(T, type)              // T : 회전의 중심 노드
    switch type:
        case LL: rightRotation(T)              // 우회전
        case LR: leftRotation(T.left)
                 balanceAVL(T, LL)             // rightRotation(T)
        case RR: leftRotation(T)               // 좌회전
        case RL: rightRotation(T.right)
                 balanceAVL(T, RR)             // leftRotation(T)
end balanceAVL()
```

AVL 트리: 구성 및 구현 (3/11)

- **AVL 트리: 구성 및 구현**
  - 균형 맞추기: LL 유형

우회전(Right Rotation)

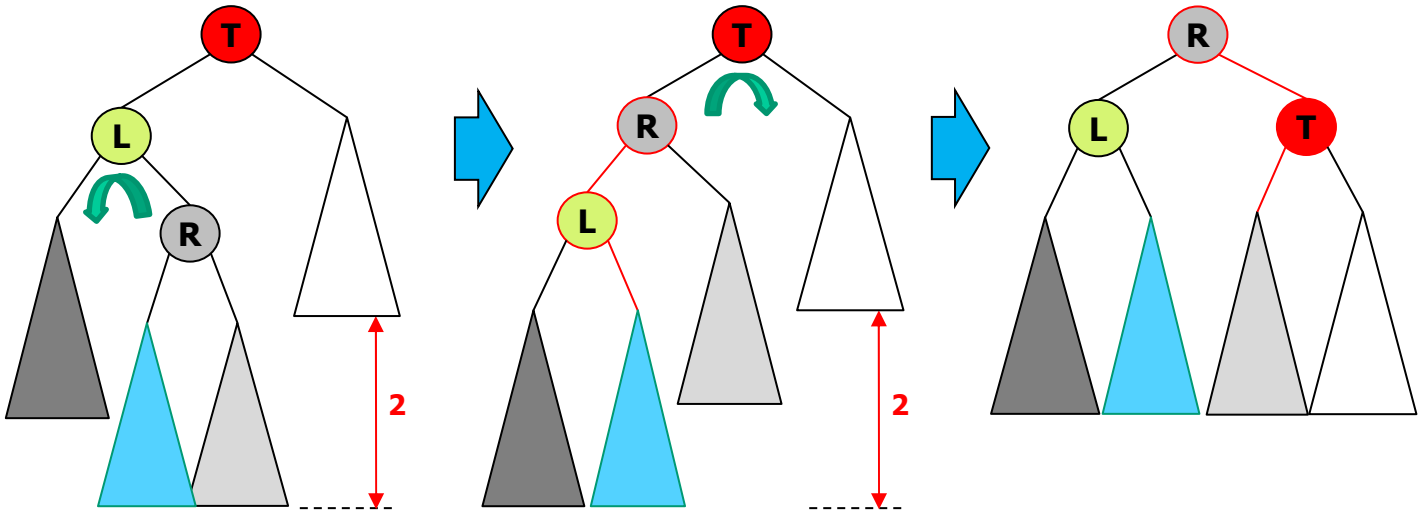



```
// AVL 트리의 우회전
rightRotation(T)                    // T : 회전의 중심 노드
    LChild ← T.Llink;
    LRChild ← LChild.Rlink;
    LChild.Rlink ← T;
    T.Llink ← LRChild;
    LChild.height ← max(LChild.Llink.height, LChild.Rlink.height) + 1;
    T.height ← max(T.Llink.height, T.Rlink.height) + 1;
end rightRotation()
```

AVL 트리: 구성 및 구현 (4/11)

- **AVL 트리:** 구성 및 구현
  - 균형 맞추기: LR 유형

좌회전 후 우회전
(타입 LL로 변환)

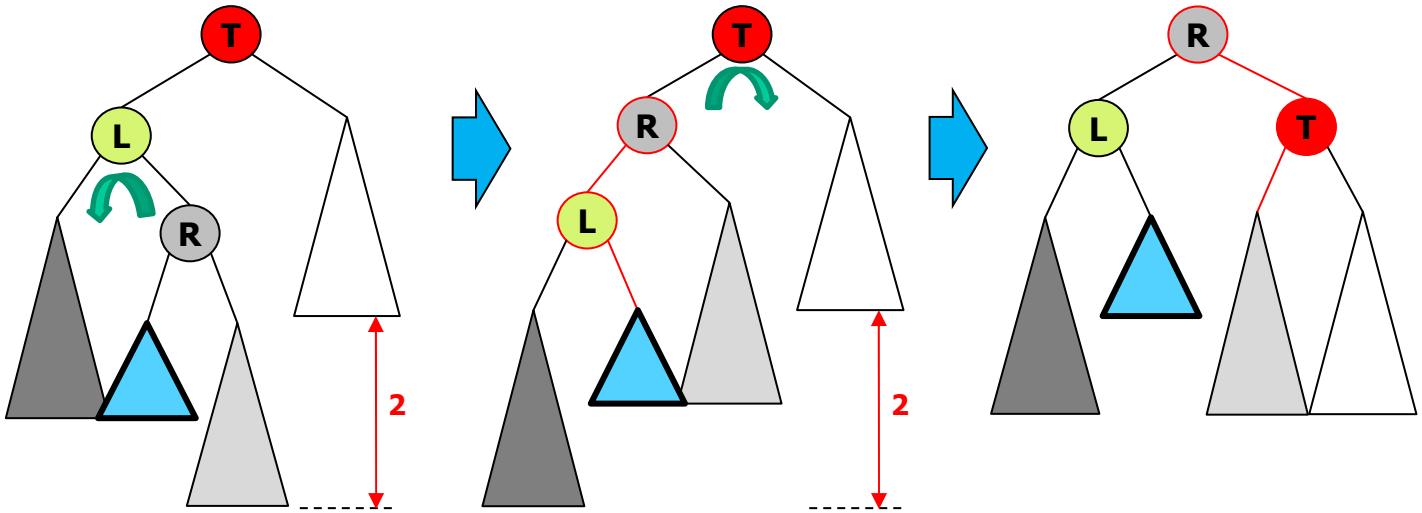


Type LR　　Type LL

AVL 트리: 구성 및 구현 (5/11)

- **AVL 트리:** 구성 및 구현
  - 균형 맞추기: LR 유형

좌회전 후 우회전
(타입 LL로 변환)

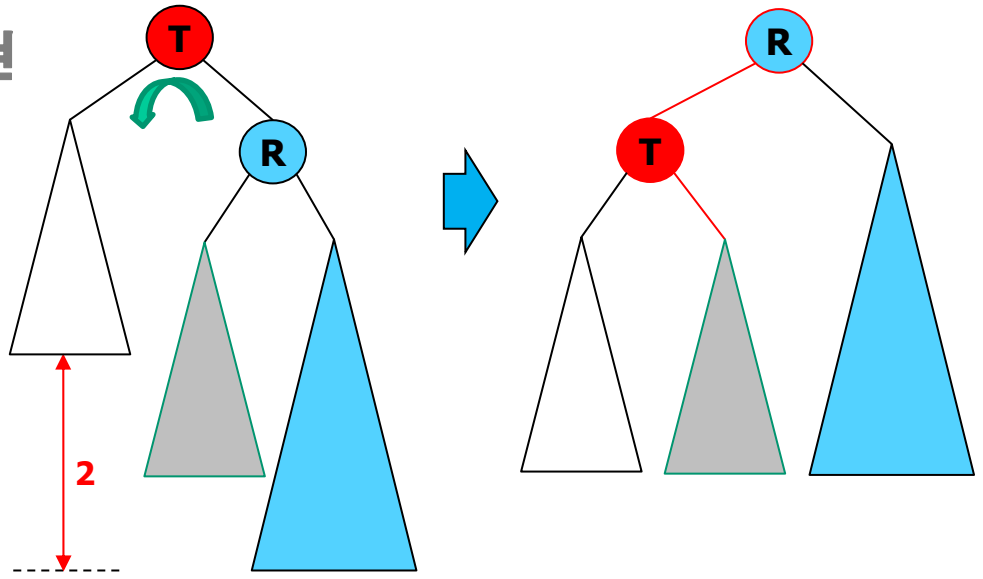


Type LR　　Type LL

AVL 트리: 구성 및 구현 (6/11)

- **AVL 트리:** 구성 및 구현
  - 균형 맞추기: RR 유형

좌회전(Left Rotation)

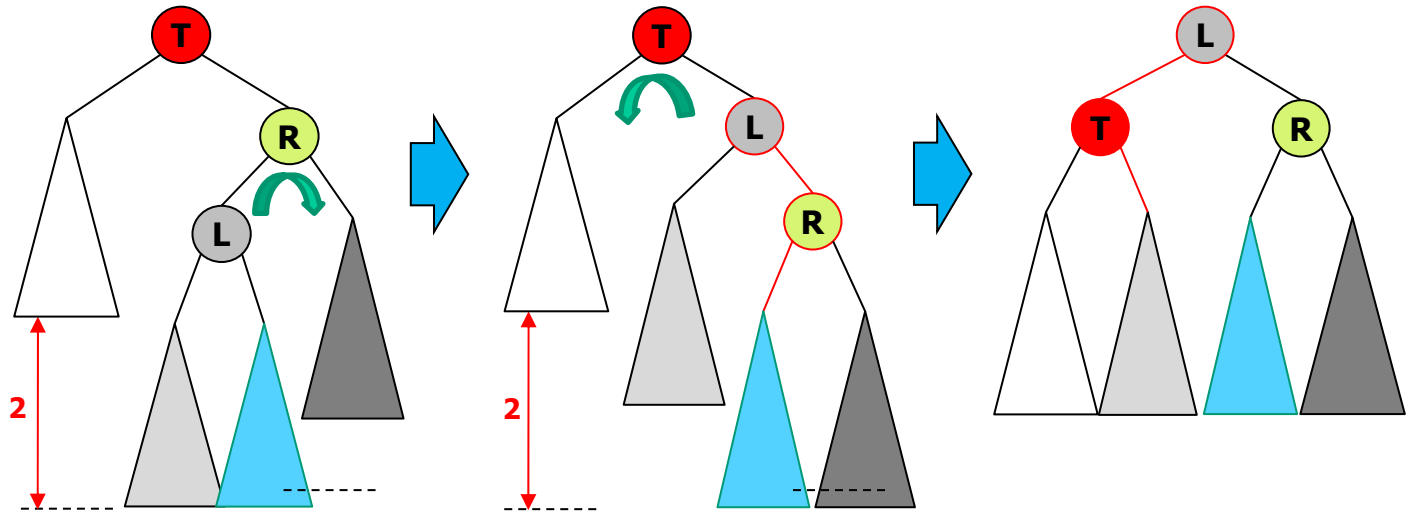



```
// AVL 트리의 좌회전
leftRotation(T)                    // T : 회전의 중심 노드
    RChild ← T.Rlink;
    RLChild ← RChild.Llink;
    RChild.Llink ← T;
    T.Rlink ← RLChild;
    RChild.height ← max(RChild.Llink.height, RChild.Rlink.height) + 1;
    T.height ← max(T.Llink.height, T.Rlink.height) + 1;
end rightRotation()
```

AVL 트리: 구성 및 구현 (7/11)

- **AVL 트리:** 구성 및 구현
  - 균형 맞추기: RL 유형

우회전 후 좌회전
(타입 RR로 변환)



Type RL　　Type RR

AVL 트리: 구성 및 구현 (8/11)

- **AVL 트리: 수선의 예**
  - 수선이 끝까지 올라갈 수도 있다.

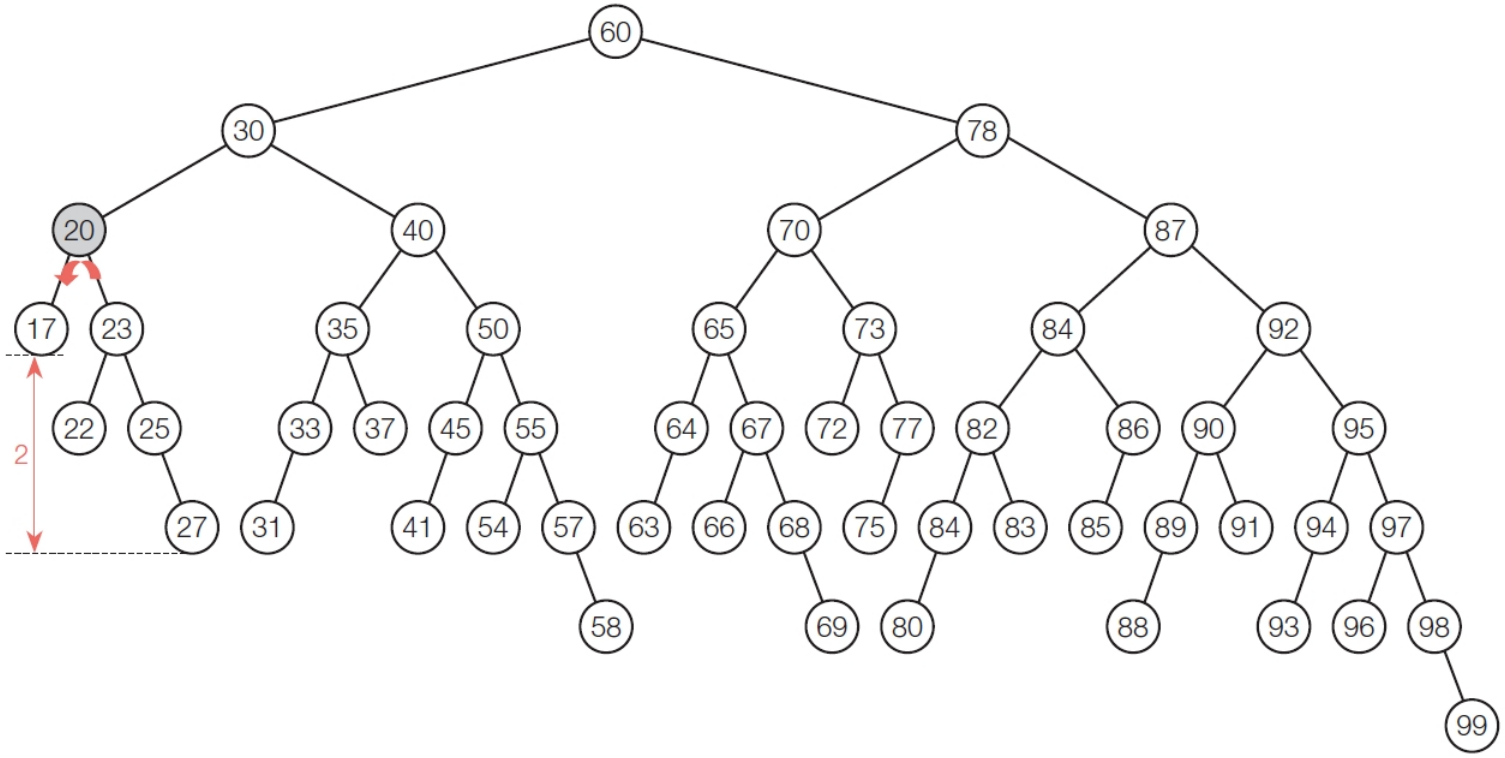


그림 11-18 RR 타입으로 균형이 깨진 예

[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

AVL 트리: 구성 및 구현 (9/11)

- **AVL 트리: 수선의 예**
  - 수선이 끝까지 올라갈 수도 있다.

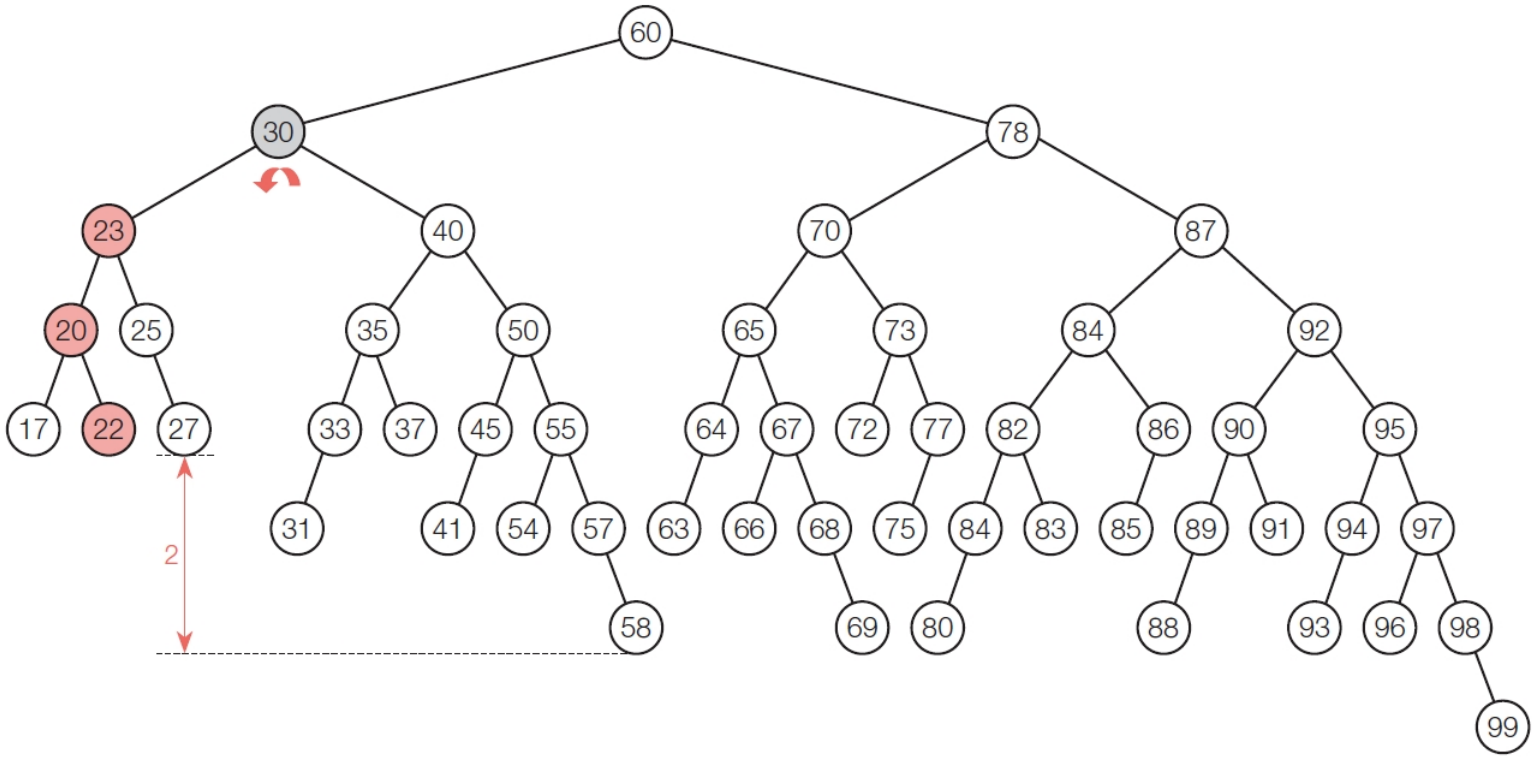


그림 11-19 한 번의 좌회전 후 상위 서브 트리에서 새롭게 균형이 깨진 상태

[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

AVL 트리: 구성 및 구현 (10/11)

- **AVL 트리: 수선의 예**
  - 수선이 끝까지 올라갈 수도 있다.

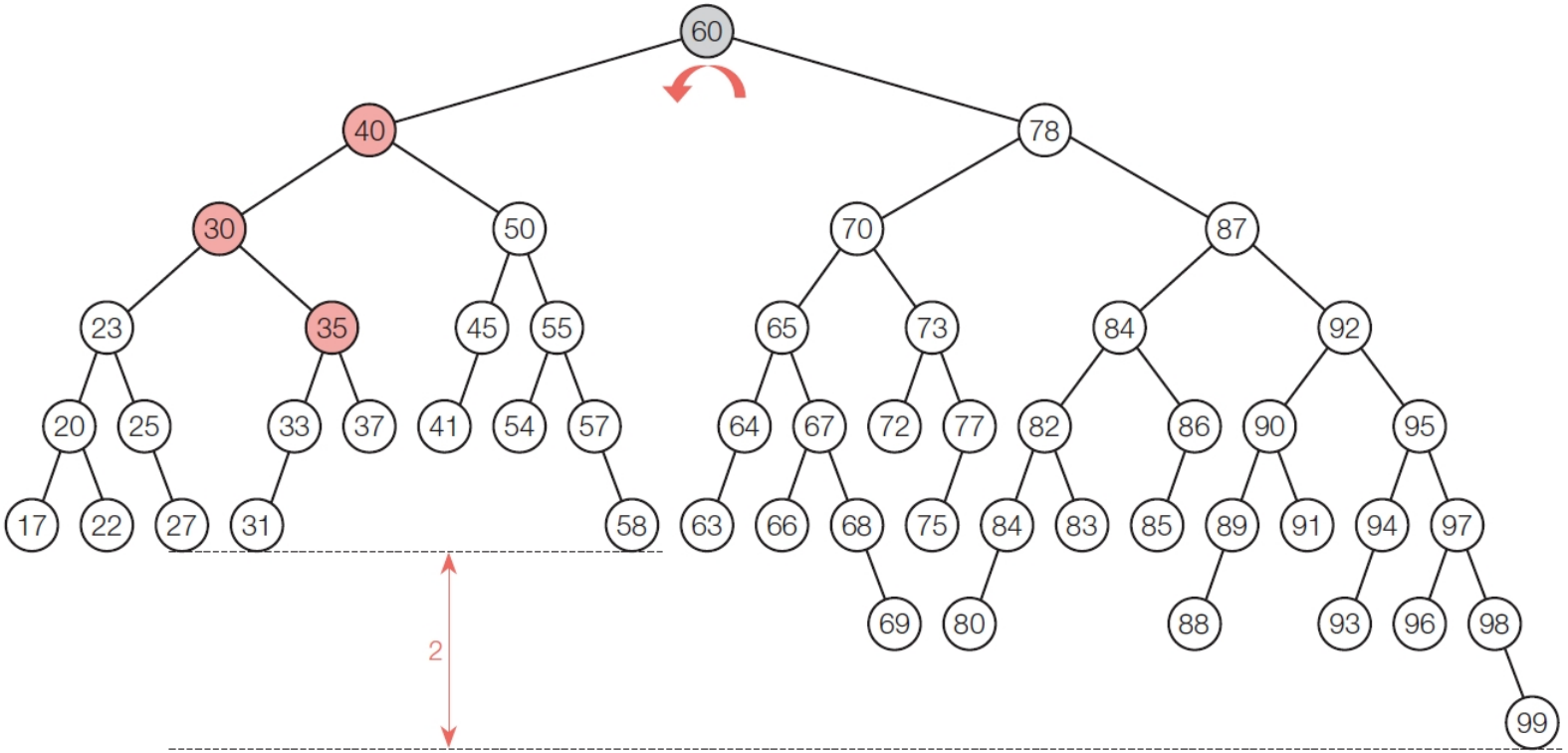


그림 11-20 두 번의 죄회전 후 상위 서브 트리에서 새롭게 균형이 깨진 상태

[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

AVL 트리: 구성 및 구현 (11/11)

- **AVL 트리: 수선의 예**
  - 수선이 끝까지 올라갈 수도 있다.

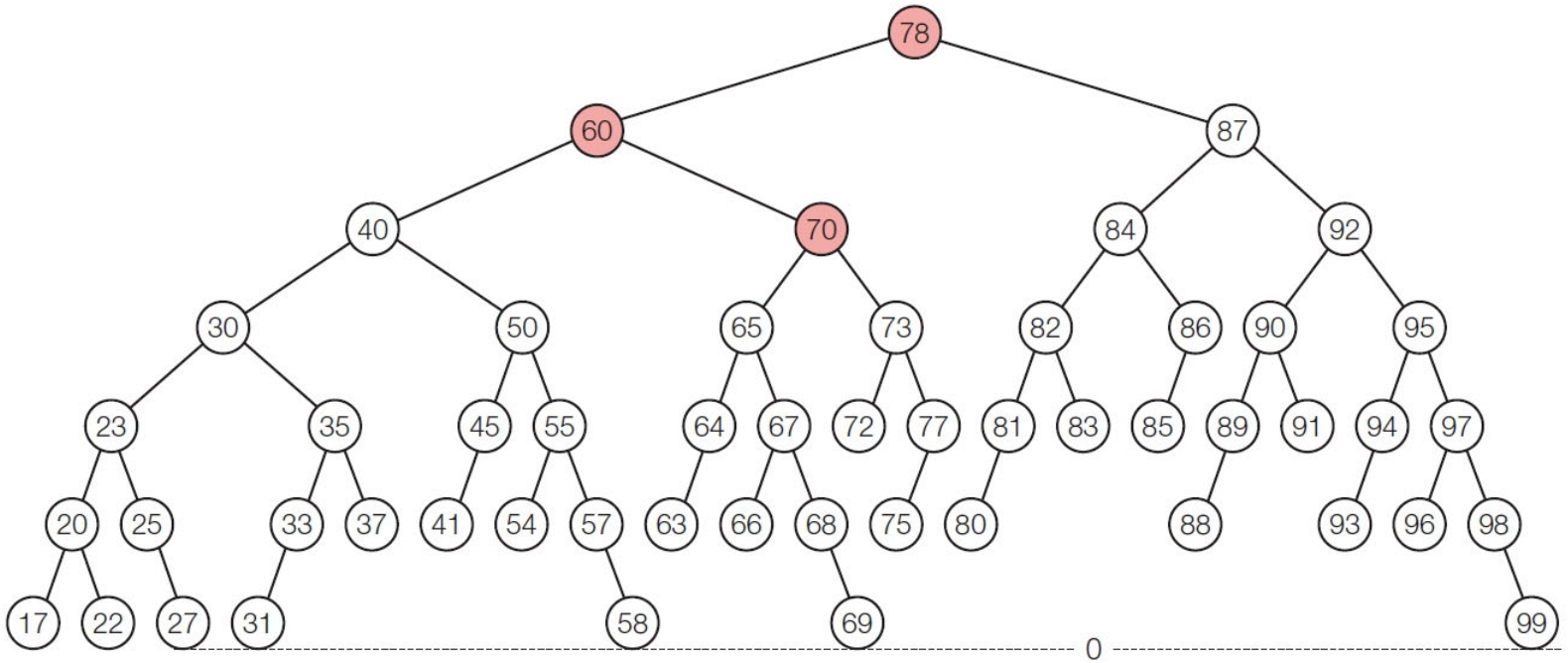


그림 11-21 세 번의 좌회전 후 균형이 해결된 상태

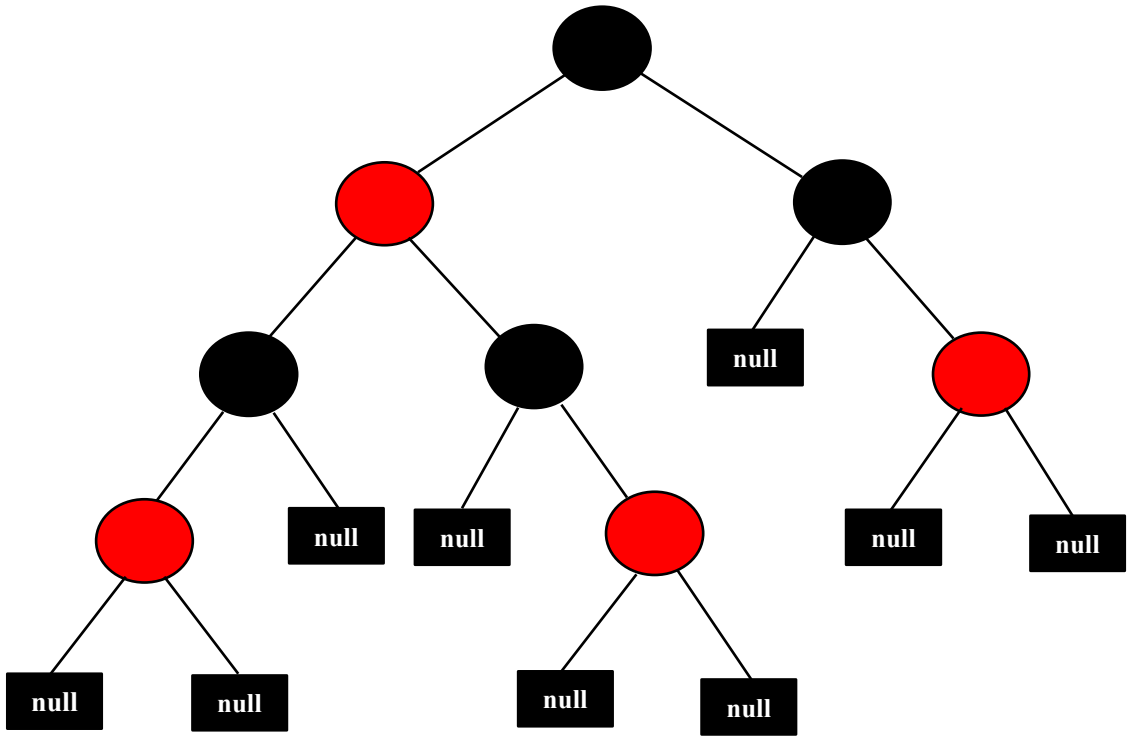
[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

균형 이진 검색 트리

레드-블랙 트리

레드-블랙 트리 (1/3)

- **레드-블랙 트리**(Red-Black Tree, RB Tree)

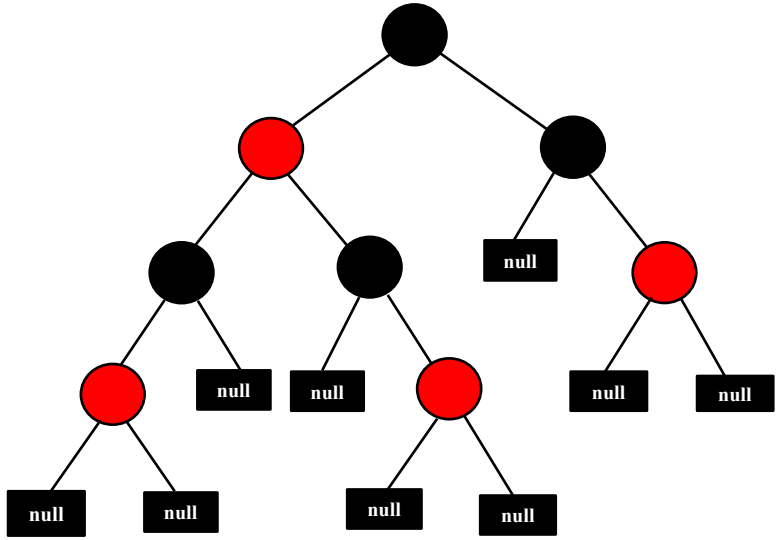


레드-블랙 트리 (2/3)

레드-블랙 트리: 특성

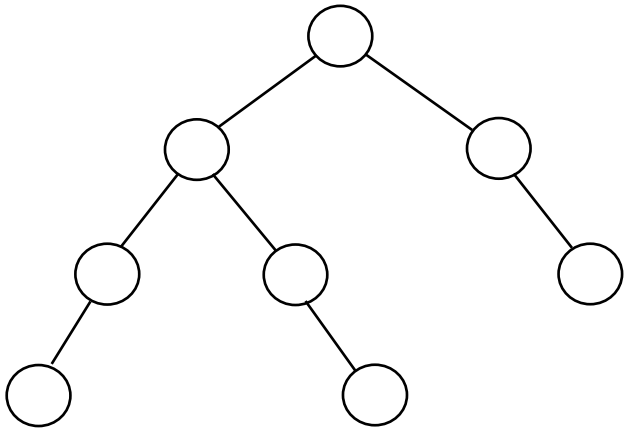
레드-블랙 트리의 특성

- 모든 null 자리에 단말 노드((leaf node)를 둔다.
  - RB-Tree에서 단말 노드는 이 null을 말한다.
- 모든 노드는 Red 또는 Black의 색을 갖는다.
  1. 루트와 모든 단말 노드는 블랙이다.
  2. 임의의 단말 노드에 이르는 경로 상에 레드 노드 두 개가 연속으로 출현하지 못한다.
  3. 임의의 단말 노드에 이르는 경로에서 만나는 블랙 노드의 수(black height)는 모두 같다.

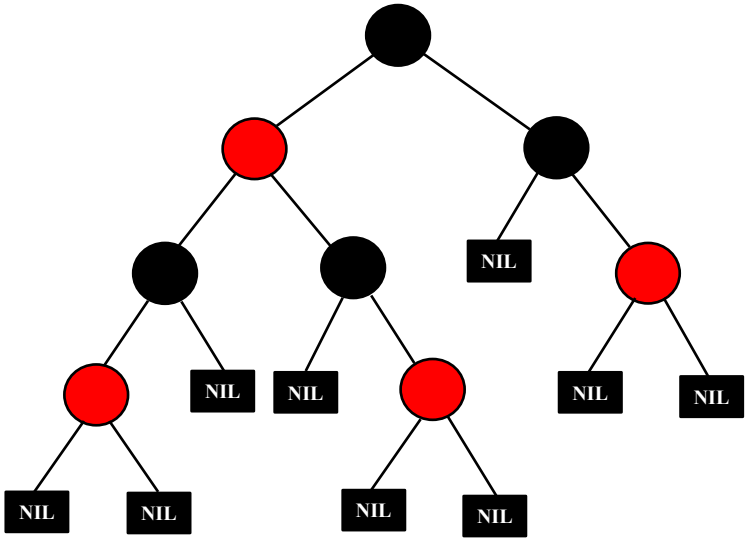


레드-블랙 트리 (3/3)

- **레드-블랙 트리: 특성**
  - 레드-블랙 트리의 예



(a) 이진 검색 트리의 예



(b) (a)를 RB-Tree로 만든 예

균형 이진 검색 트리

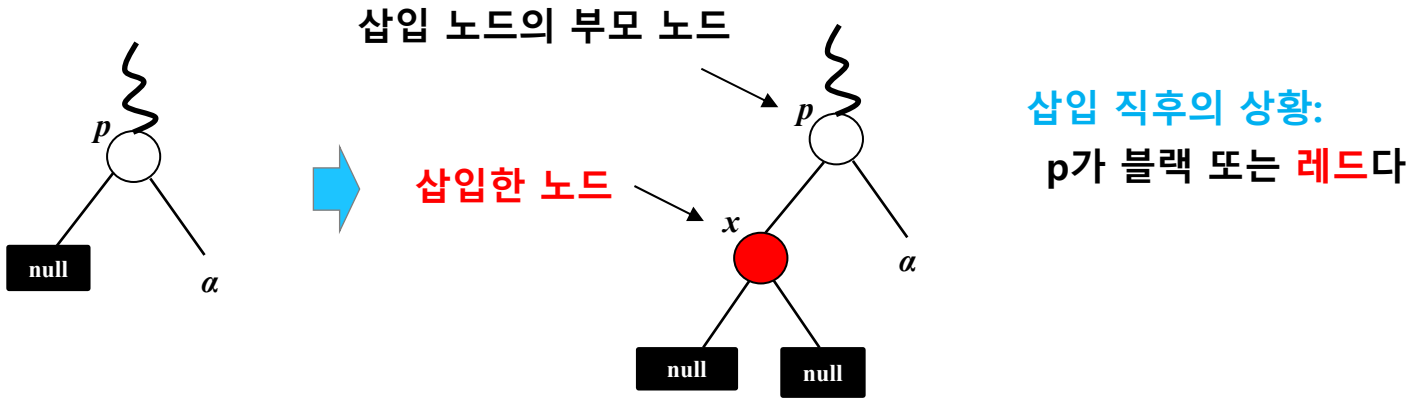
레드-블랙 트리: 구성 및 구현

레드-블랙 트리: 구성 및 구현 (1/8)

레드-블랙 트리: 구성 및 구현

레드-블랙 트리의 삽입 #1

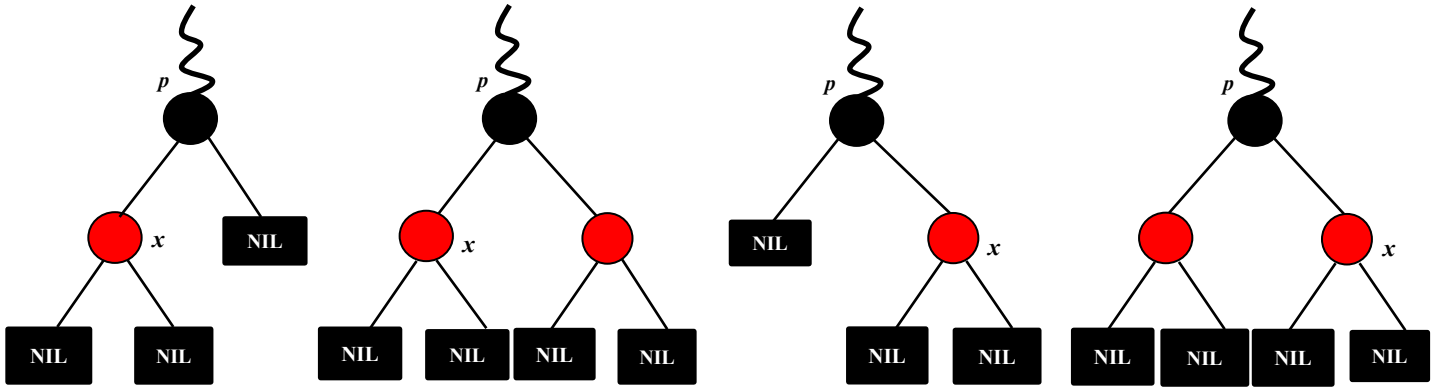
- 일반적인 BST의 삽입 작업 후 삽입 노드에 레드를 칠하고,
- 삽입한 노드의 좌우에 null 리프를 달아준다.



삽입 직후의 상황:
p가 블랙 또는 레드다

레드-블랙 트리: 구성 및 구현 (2/8)

- **레드-블랙 트리: 구성 및 구현**
  - 레드-블랙 트리의 삽입 #2
    - 삽입 직후의 상황: x 의 부모 p 가 블랙 또는 레드다.
    - 만약 p 가 블랙: RB-Tree 특성을 모두 만족. 완료!!!



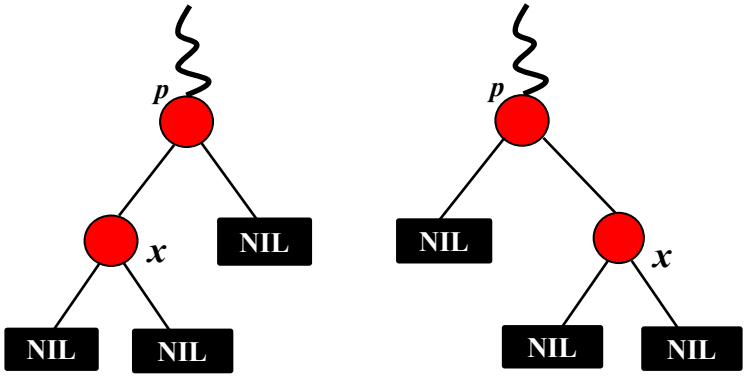
가능한 모양은 이 4가지 뿐이다

레드-블랙 트리: 구성 및 구현 (3/8)

레드-블랙 트리: 구성 및 구현

레드-블랙 트리의 삽입 #3

- 만약 p 가 레드
  - RB 특성 ③ 이 깨졌다. → 수선 (다음 페이지 이후)
  - 임의의 단말 노드에 이르는 경로 상에 레드 노드 두 개가 연속으로 출현하지 못한다.



가능한 모양은 이 2가지 뿐이다

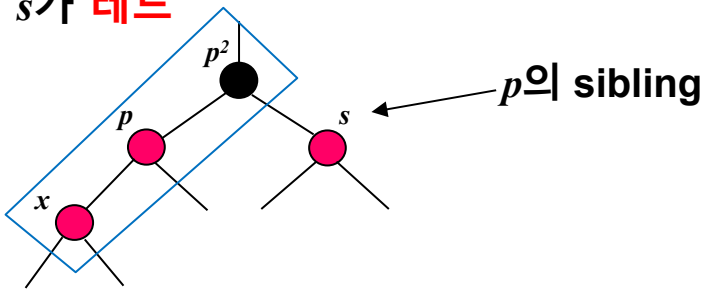
레드-블랙 트리: 구성 및 구현 (4/8)

레드-블랙 트리: 구성 및 구현

레드-블랙 트리의 삽입 #4

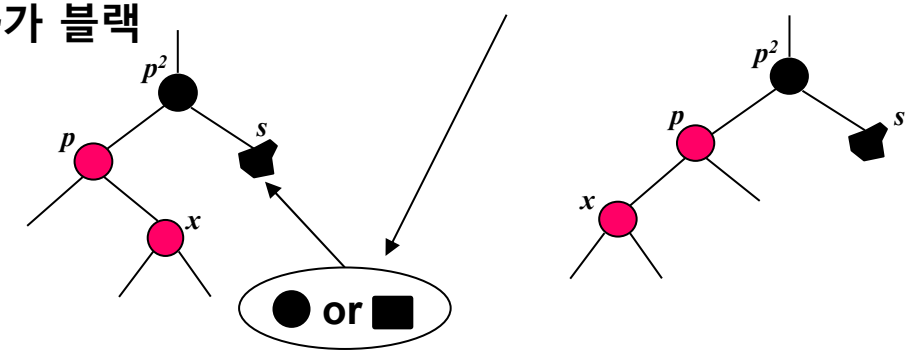
- 만약 p 가 레드
  - p 의 형제sibling 노드 s 에 따라 두 가지로 나눈다.

Case 1: *s*가 레드



✓ 질문: 여기서 ● 가 가능한가?

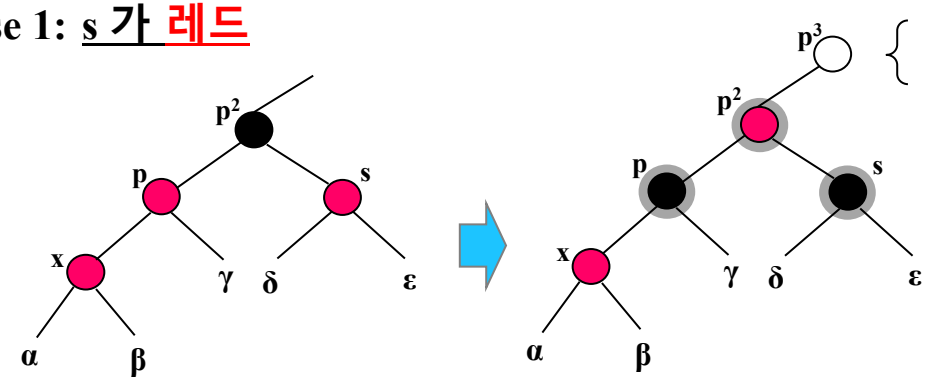
Case 2: *s*가 블랙



레드-블랙 트리: 구성 및 구현 (5/8)

- **레드-블랙 트리:** 구성 및 구현
  - 레드-블랙 트리의 삽입 #5

Case 1: s 가 레드



P³ 가 블랙이면 완료
P³가 레드이면 p²가 새 *x* : Recursive!

: 색이 바뀐 노드

p 와 s 를 블랙으로 바꾸고,

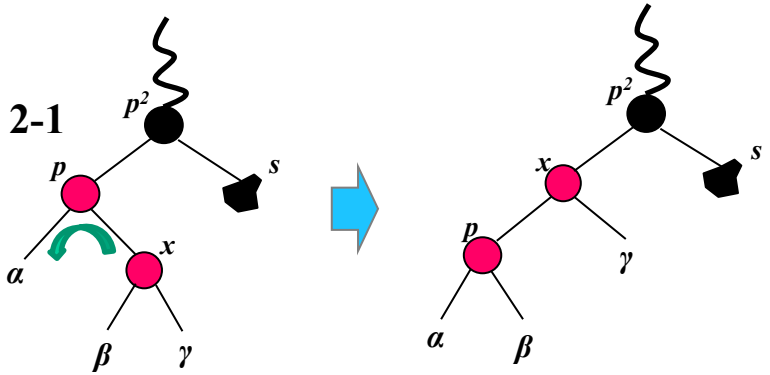
p^2을 레드로 바꾼다.

레드-블랙 트리: 구성 및 구현 (6/8)

- **레드-블랙 트리:** 구성 및 구현
  - 레드-블랙 트리의 삽입 #6

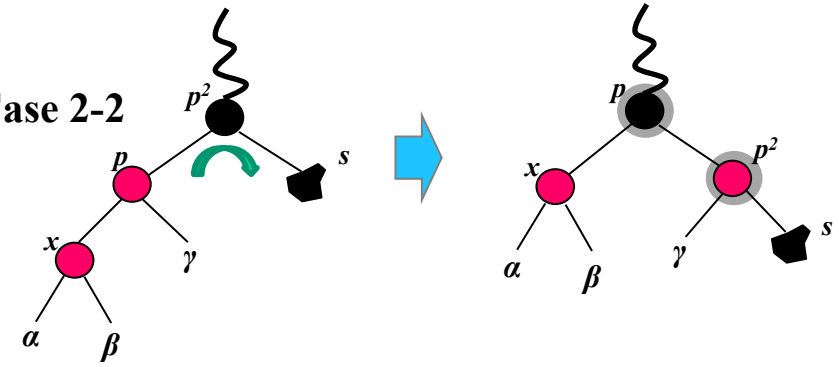
Case 2: s 가 블랙

Case 2-1



Case 2-2로 변환

Case 2-2



우회전하고, p와 p^2의 색을 바꾼다. 수선 끝.

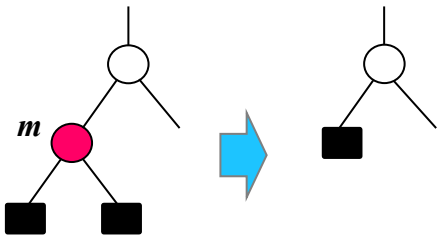
레드-블랙 트리: 구성 및 구현 (7/8)

● 레드-블랙 트리: 구성 및 구현

❍ 레드-블랙 트리의 삭제 #1

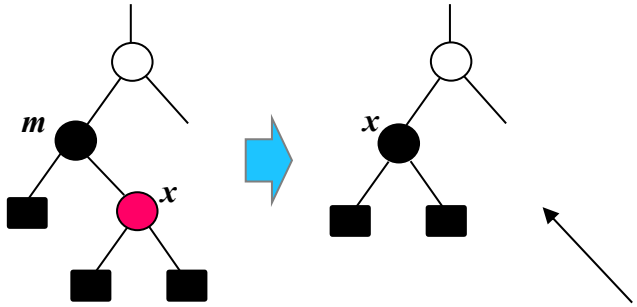
- BST 의 삭제 작업 중 Case 1과 2만 고려하면 된다.

✓ 질문: Case 3은 왜 고려하지 않아도 되는가?



삭제 노드(m)가 레드이면 문제없다

(m은 반드시 자식이 없다)



m 삭제 후

x의 색을 블랙으로 바꾸어준다

삭제 노드가 블랙이라도

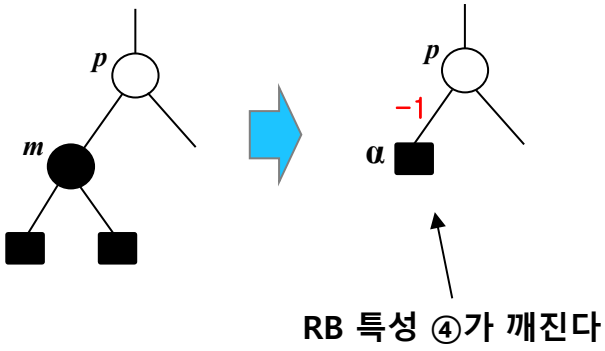
m이 자식을 가지면(유일하고 반드시 레드다) 문제없다

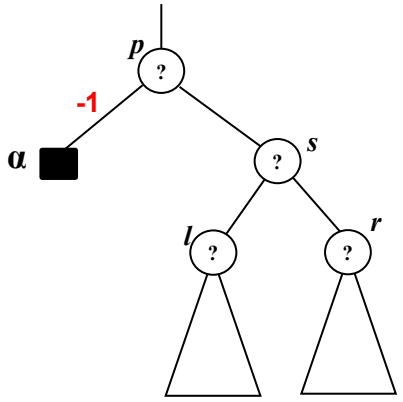
레드-블랙 트리: 구성 및 구현 (8/8)

- **레드-블랙 트리: 구성 및 구현**
  - 레드-블랙 트리의 삭제 #2

삭제 노드(m)가 블랙이고

자식이 없을 때만 문제 발생





x의 주변 상황에 따라 처리 방법이 달라진다

이 강좌에서는 여기까지만.

균형 다진 검색 트리

백문이불여일타(百聞而不如一打)

- 이진 검색 트리
- 균형 이진 검색 트리
- **균형 다진 검색 트리**
  - 2-3-4 트리
  - B-트리

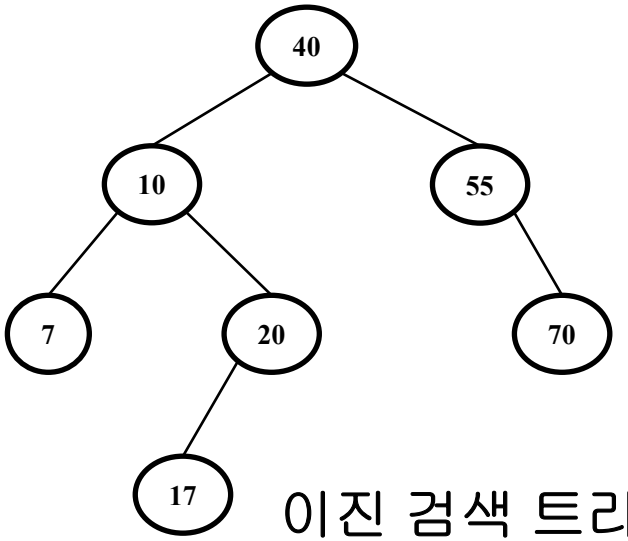
균형 다진 검색 트리 (1/3)

- **균형 검색 트리**(Binary Search Tree)
  - 다진 검색 트리: 2-3-4트리, B-트리

[이미지 출처: 문병로, "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 한빛아카데미, 2022.]

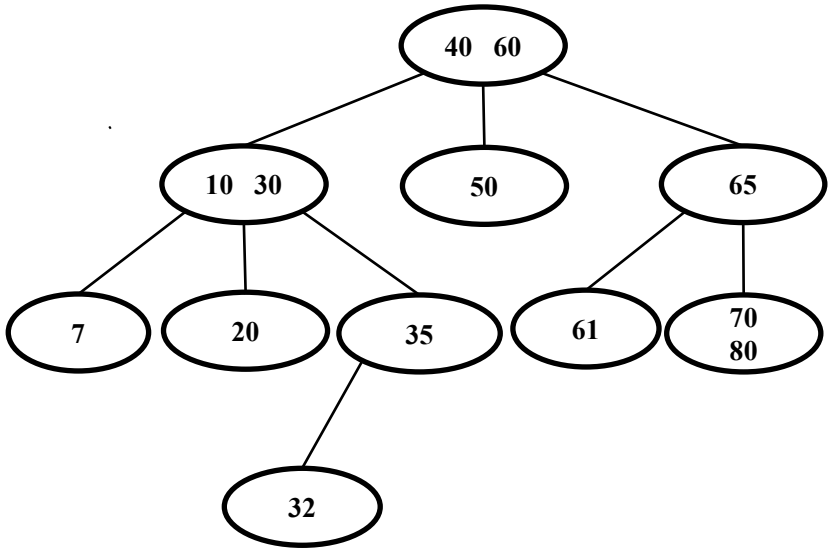
균형 다진 검색 트리 (2/3)

- **균형 검색 트리:** 다진 검색 트리
  - 다진 검색 트리: K-진 검색 트리



이진 검색 트리

K=2, 최대 2개 분기

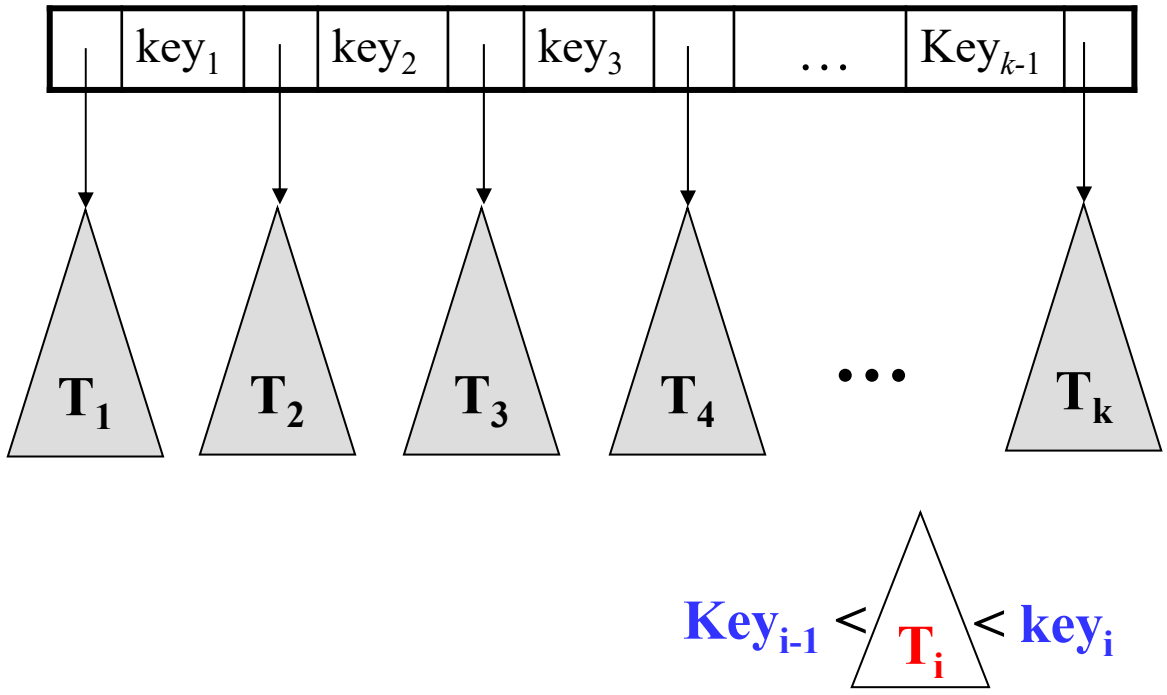


삼진 검색 트리

K=3, 최대 3개 분기

균형 다진 검색 트리 (3/3)

- **균형 검색 트리:** 다진 검색 트리
  - 다진 검색 트리: K-진 검색 트리



균형 다진 검색 트리

B-트리

B-트리 (1/12)

B-트리(B-Tree)

B-트리의 환경

- 디스크의 접근 단위는 블록(페이지)
  - 디스크에 한 번 접근하는 시간은 수십만 명령어의 처리 시간과 맞먹는다
  - 검색 트리가 디스크에 저장되어 있다면 트리의 높이를 최소화하는 것이 유리하다.
- B-트리는 K-진 검색 트리가 균형을 유지하도록 하여 최악의 경우 디스크 접근 횟수를 줄인 것이다.

B-트리의 성질

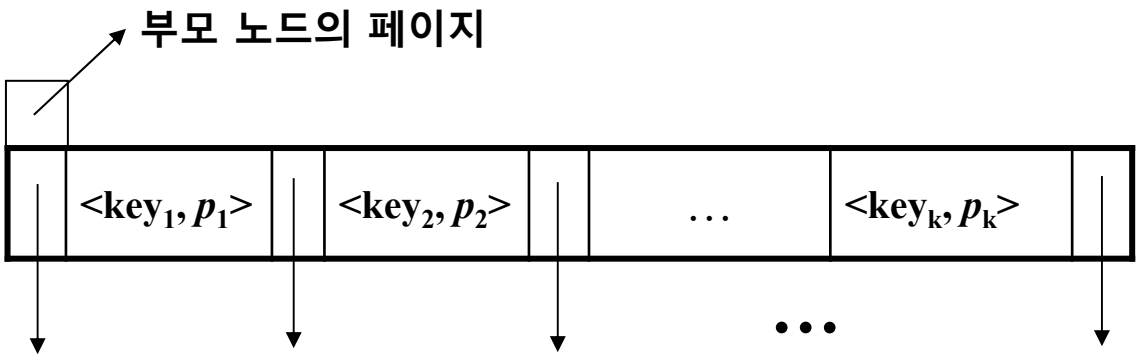
- B-트리는 균형 (K+1)-진 검색 트리로 다음의 성질을 만족한다.

① root를 제외한 모든 노드는 $\lceil K/2 \rceil \sim K$ 개의 키를 갖는다.

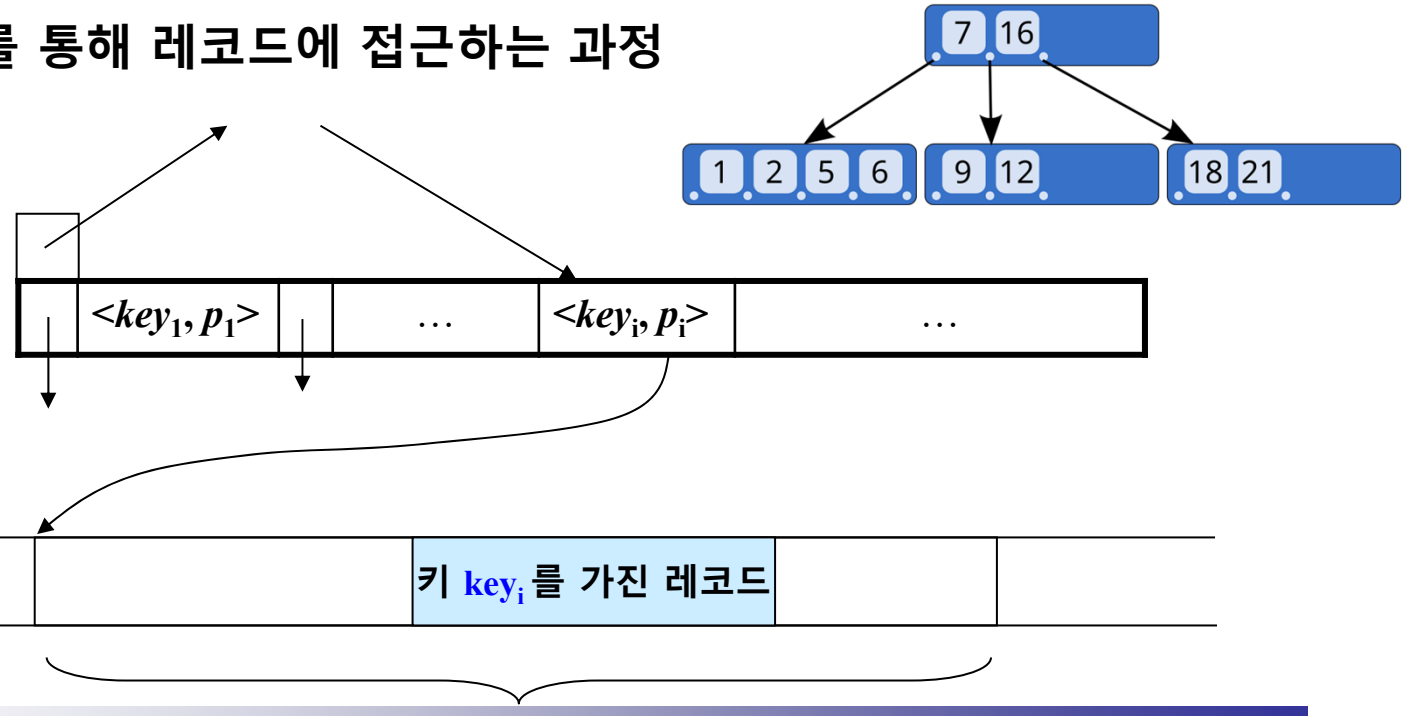
② 모든 단말 노드는 같은 깊이를 가진다

B-트리 (2/12)

- **B-트리: 노드 구조**
  - B-트리의 노드 구조



  - B-Tree를 통해 레코드에 접근하는 과정



B-트리 (3/12)

- **B-트리: 삽입 알고리즘**
  - B-트리에서의 삽입 알고리즘

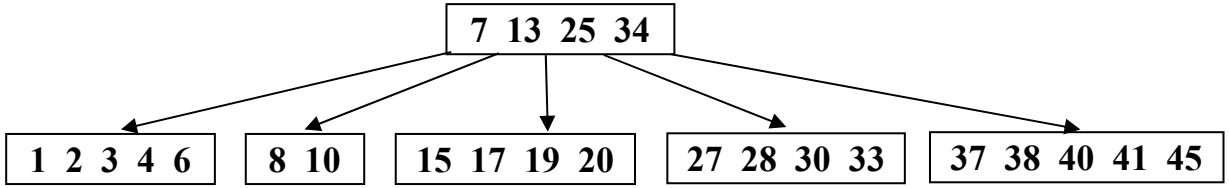
```
BTreeInsert(T, x)          ▷ t : 트리의 루트 노드   ▷ x : 삽입하고자 하는 키
{
        x 를 삽입할 리프 노드 r 을 찾는다;
        x 를 r 에 삽입한다;
        if (r 에 overflow 발생) then clearOverflow(r);
}
clearOverflow(r)
{
    if (r 의 형제 노드 중 여유가 있는 노드가 있음) then { r 의 남는 키를 넘긴다 };
    else {
            r 을 둘로 분할하고 가운데 키를 부모 노드로 넘긴다;
            if (부모 노드 p 에 overflow 발생) then clearOverflow(p);
    }
}
```

B-트리 (4/12)

- **B-트리: 삽입 알고리즘**
  - B-트리에서의 삽입 동작 과정

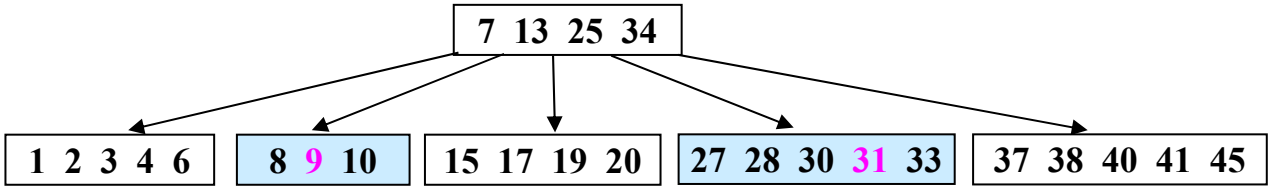
K = 5 인 경우

1



9, 31 삽입

2



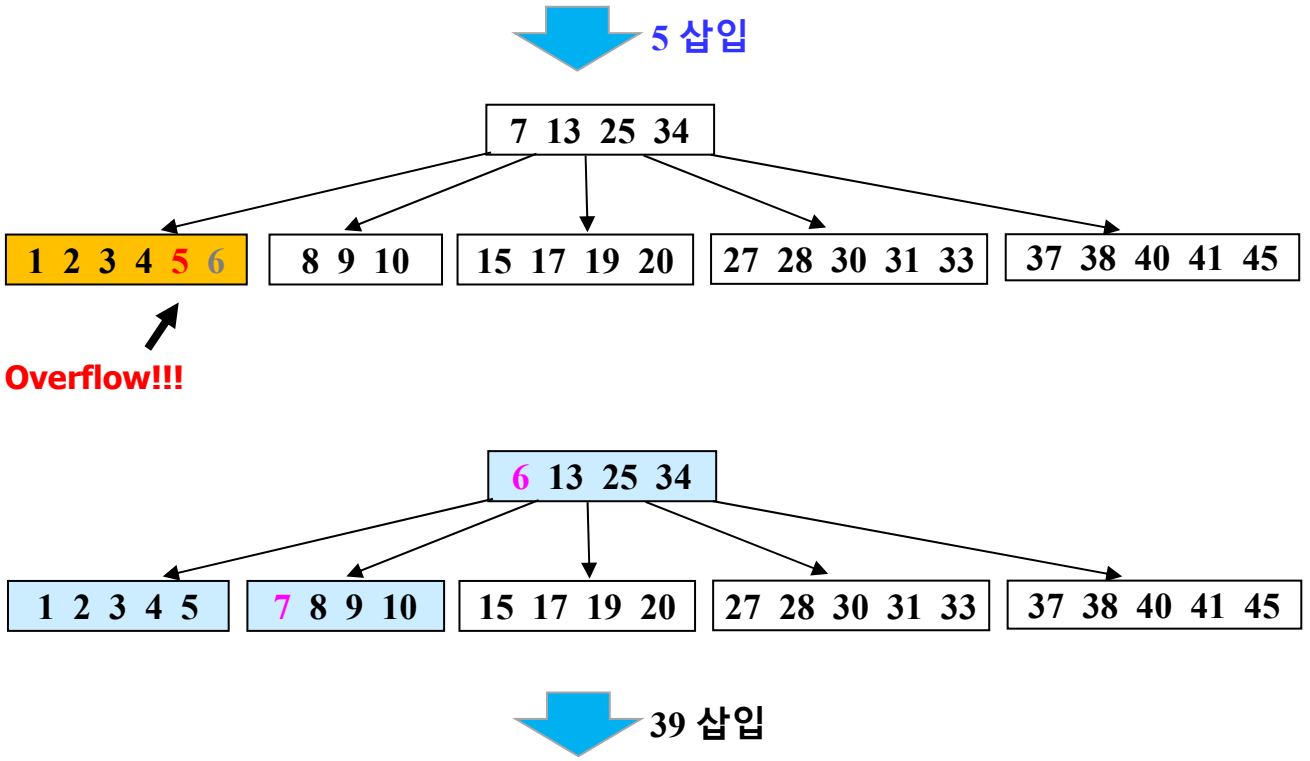
5 삽입

B-트리 (5/12)

- **B-트리:** 삽입 알고리즘
  - B-트리에서의 삽입 동작 과정

K = 5 인 경우

3

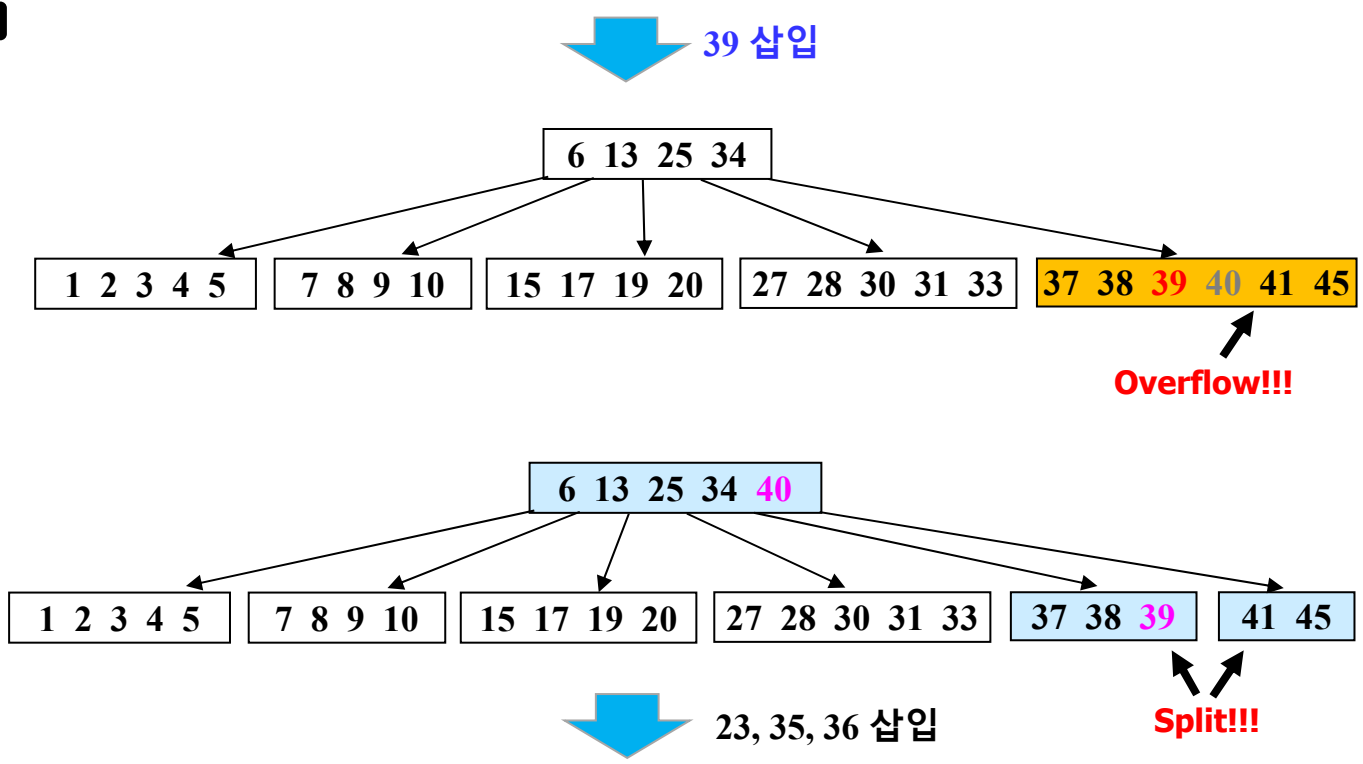


B-트리 (6/12)

- **B-트리: 삽입 알고리즘**
  - B-트리에서의 삽입 동작 과정

K = 5 인 경우

4

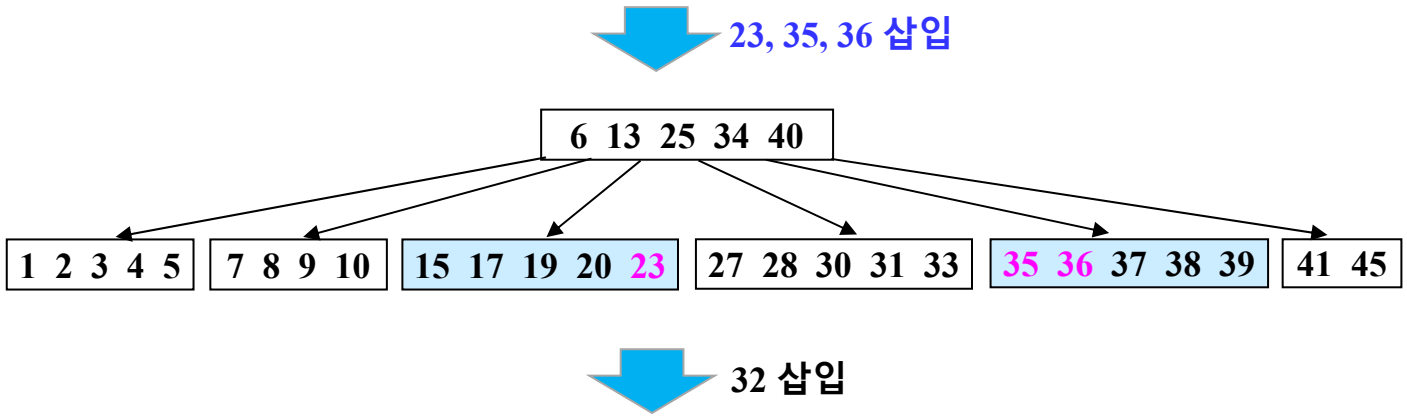


B-트리 (7/12)

- **B-트리: 삽입 알고리즘**
  - B-트리에서의 삽입 동작 과정

K = 5 인 경우

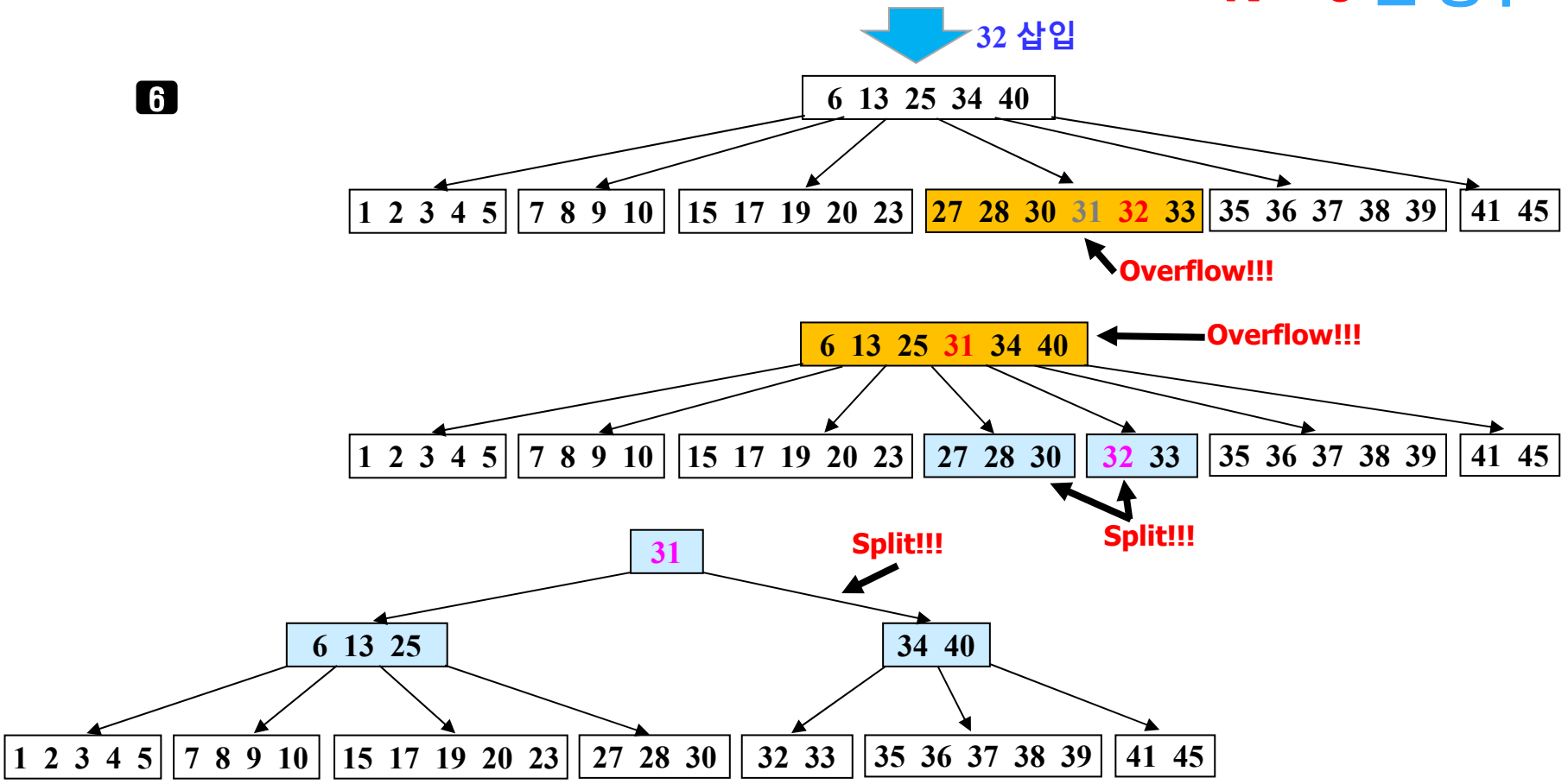
5



B-트리 (8/12)

- **B-트리:** 삽입 알고리즘
  - B-트리에서의 삽입 동작 과정

K = 5 인 경우



B-트리 (9/12)

B-트리: 삭제 알고리즘

B-트리에서의 삭제 알고리즘

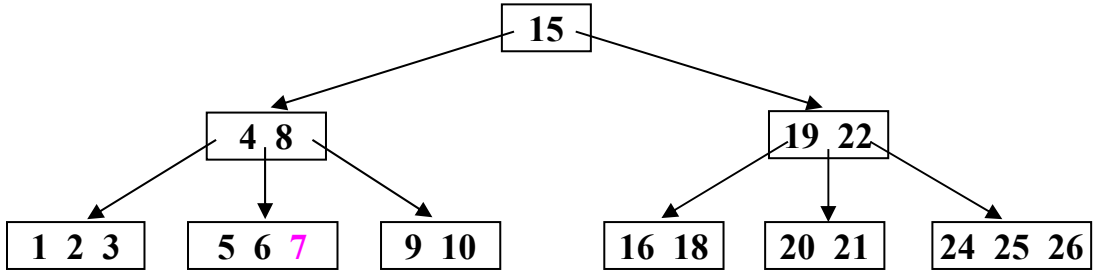
```
BTreeDelete(T, x, v)        ▷ t : 트리의 루트 노드   ▷ x : 삭제하고자 하는 키 ▷ v : x를 갖고 있는 노드
{
        if (v 가 단말 노드 아님) then {
                x 의 직후 원소  y 를 가진 단말 노드를 찾는다;
                x 와  y를 맞바꾼다;
        }
        단말 노드에서 x 를 제거하고 이 단말 노드를  r 이라 한다;
        if (r 에서 underflow 발생) then clearUnderflow(r);
}

clearUnderflow(r)
{
        if ( r 의 형제 노드 중 키를 하나 내놓을 수 있는 여분을 가진 노드가 있음) then {
                r 이 키를 넘겨받는다;
        }
        else {
                r 의 형제 노드와 r 을 합병한다;
                if (부모 노드  p 에 underflow 발생) then clearUnderflow(p);
        }
}
```

B-트리 (10/12)

- **B-트리:** 삭제 알고리즘
  - B-트리에서의 삭제 동작 과정

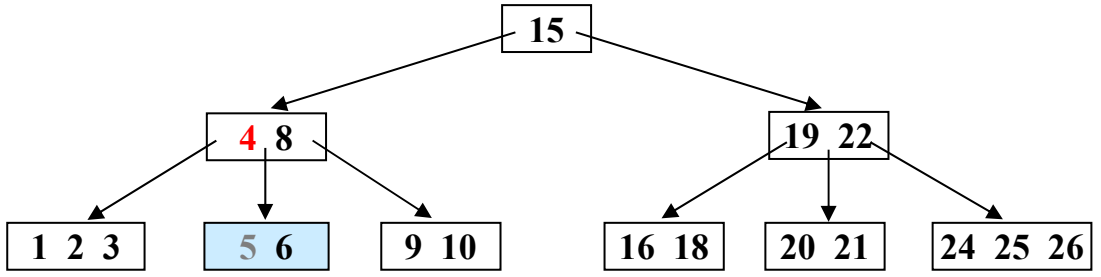
K = 5 인 경우

1



7 삭제

2



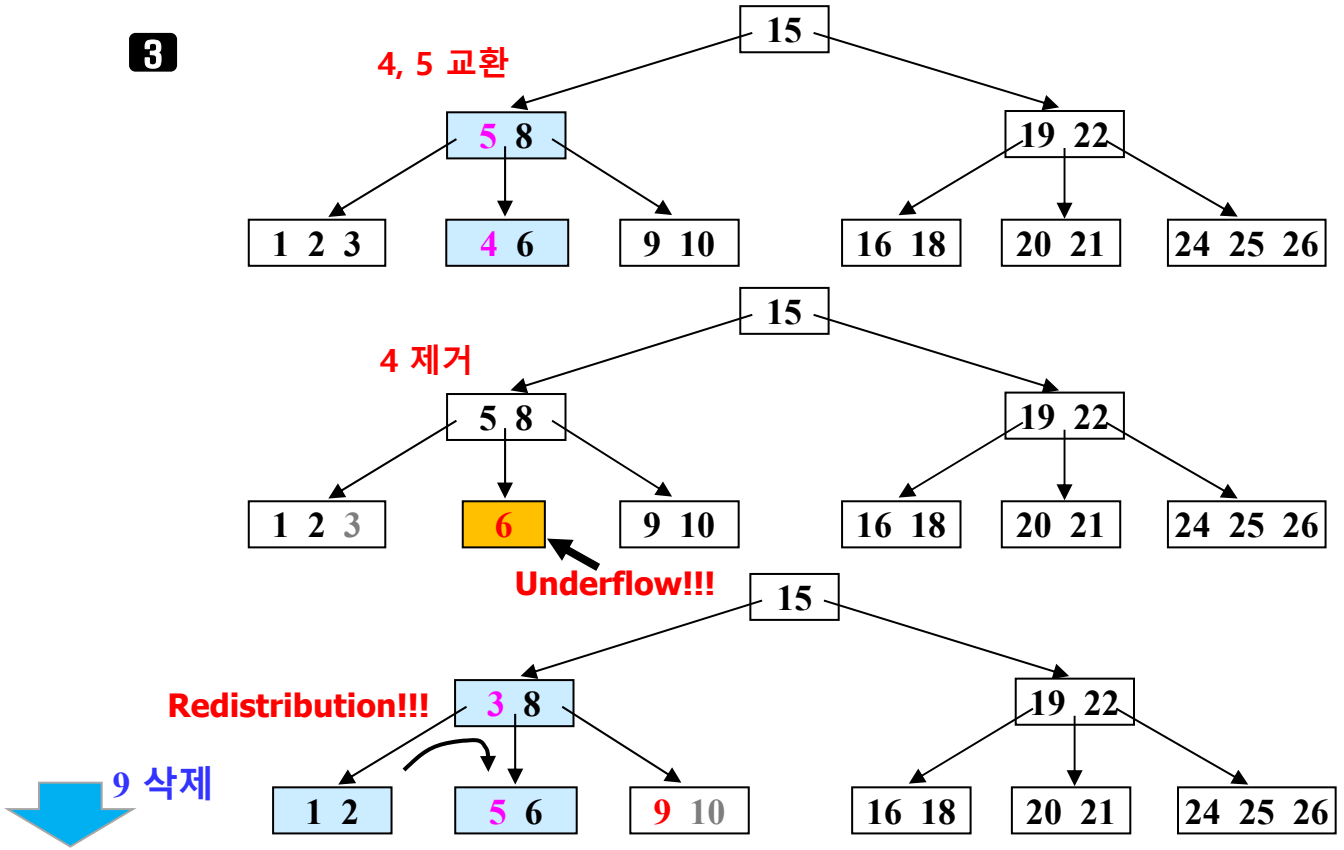
4 삭제

B-트리 (11/12)

- **B-트리:** 삭제 알고리즘
  - B-트리에서의 삭제 동작 과정

K = 5 인 경우

3



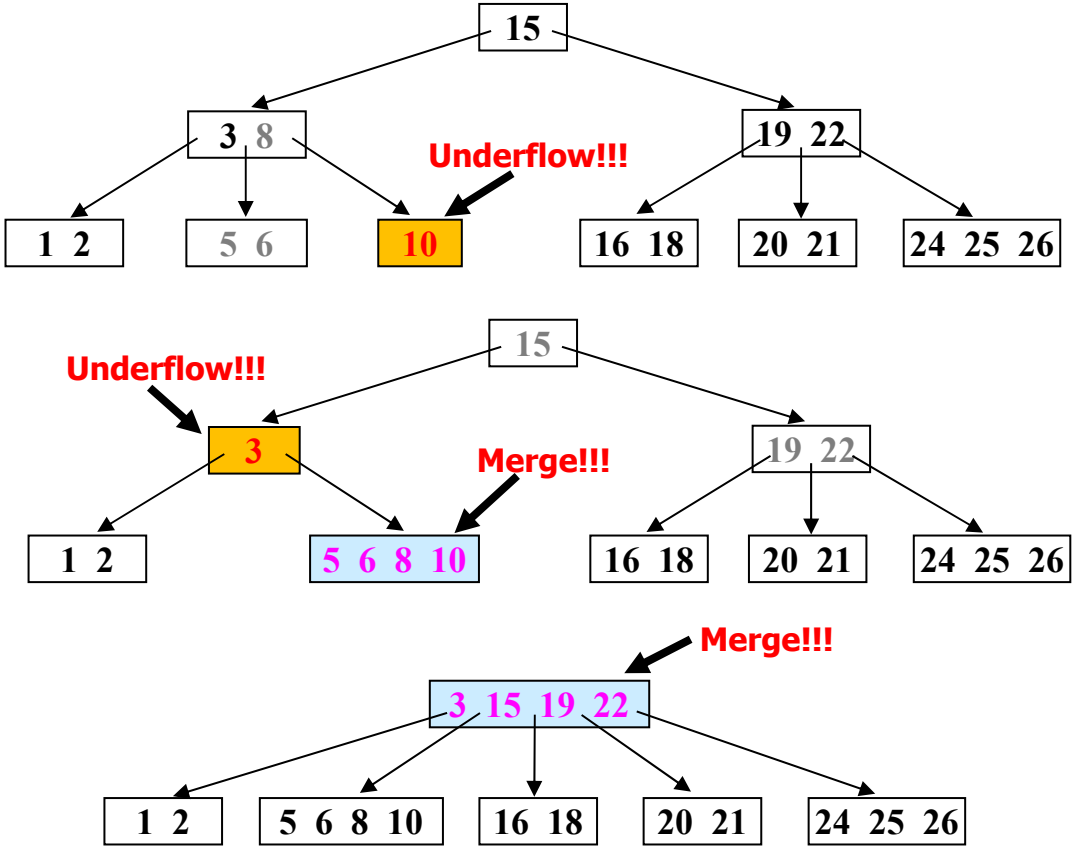
B-트리 (12/12)

- **B-트리:** 삭제 알고리즘
  - B-트리에서의 삭제 동작 과정

9 삭제

K = 5 인 경우

4



참고문헌

[1] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛미디어, 2022.

[2] "C++로 구현하는 자료구조와 알고리즘(2판)", Michael T. Goodrich, 김유성 외 2인 번역, 한빛아카데미, 2020.

[3] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.

[4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(3판), 개정판, 한빛아카데미, 2024.

[5] "코딩 테스트를 위한 자료 구조와 알고리즘 with C++", John Carey 외 2인, 황선규 역, 길벗, 2020.

[6] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.

[7] "SW Expert Academy", SAMSUNG, 2025 of viewing the site, https://swexpertacademy.com/.

[8] "BAEKJOON", (BOJ) BaekJoon Online Judge, 2025 of viewing the site, https://www.acmicpc.net/.

[9] "programmers", grepp, 2025 of viewing the site, https://programmers.co.kr/.

[10] "goormlevel", goorm, 2025 of viewing the siteh, https://level.goorm.io/