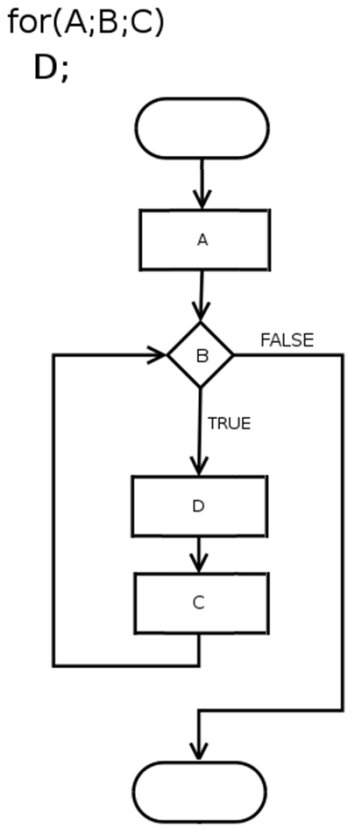
Clickseo.com

자료구조 & 알고리즘

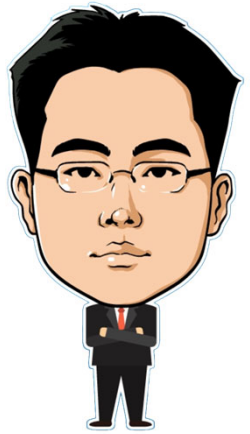


탐색 트리
(Search Tree)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차

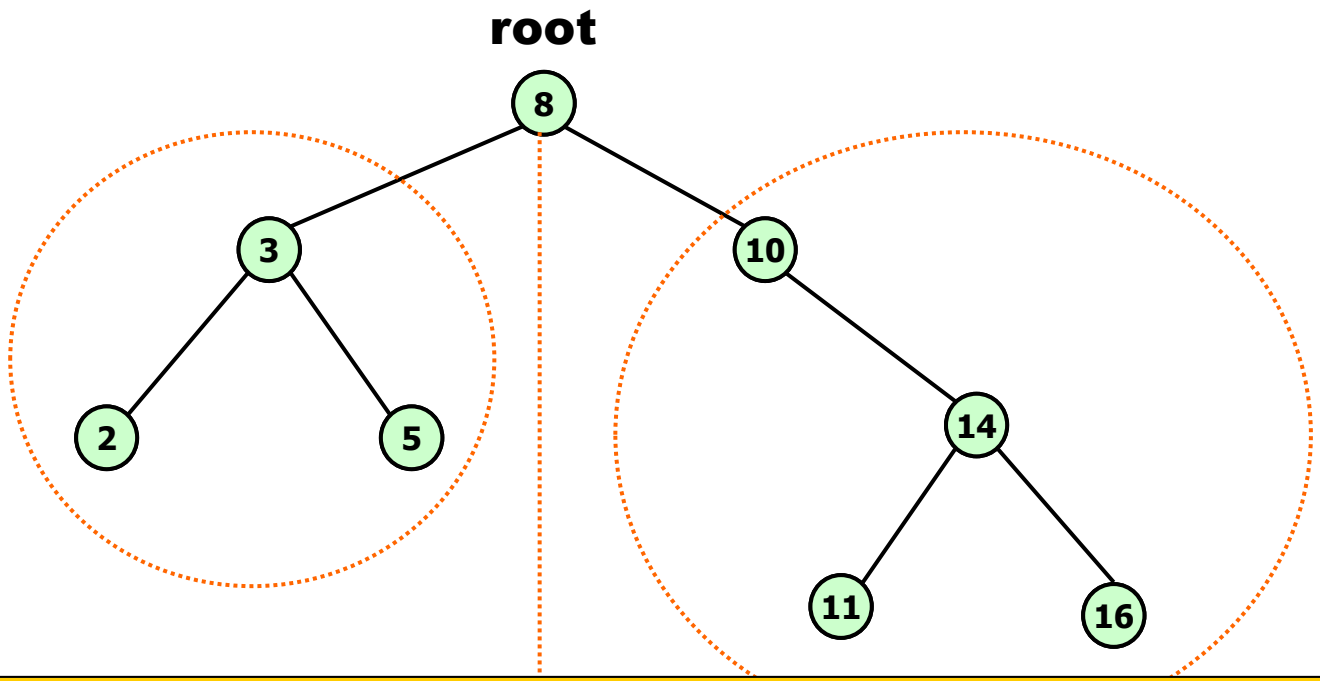
- 이진 탐색 트리
- 균형 탐색 트리

이진 탐색 트리

- 이진 탐색 트리
  - 이진 탐색 트리 연산
- 균형 탐색 트리

이진 탐색 트리 (1/3)

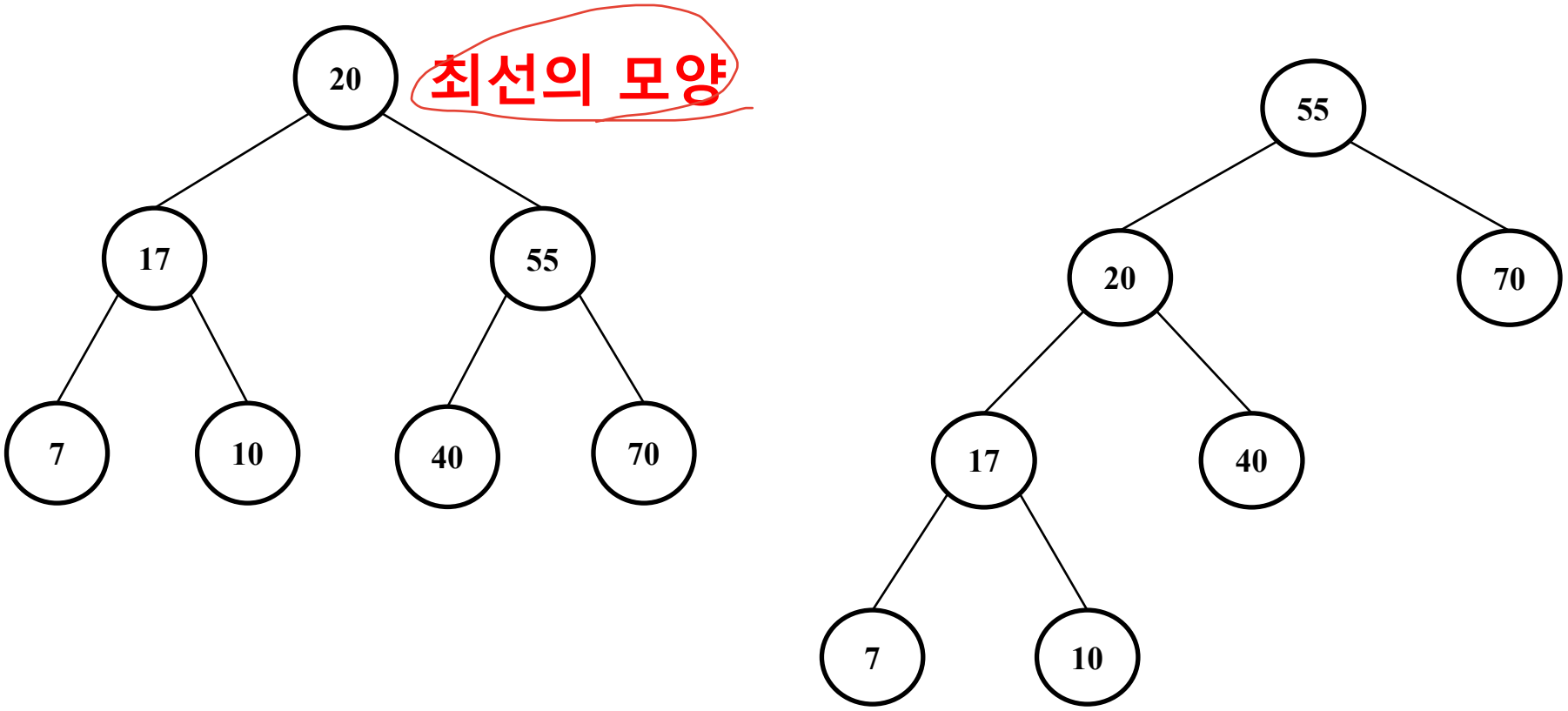
- **이진 탐색 트리**(Binary Search Tree)
  - 모든 노드는 서로 다른 키를 갖는다(유일한 키 값).
    - 각 노드는 최대 2개의 자식을 갖는다.



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

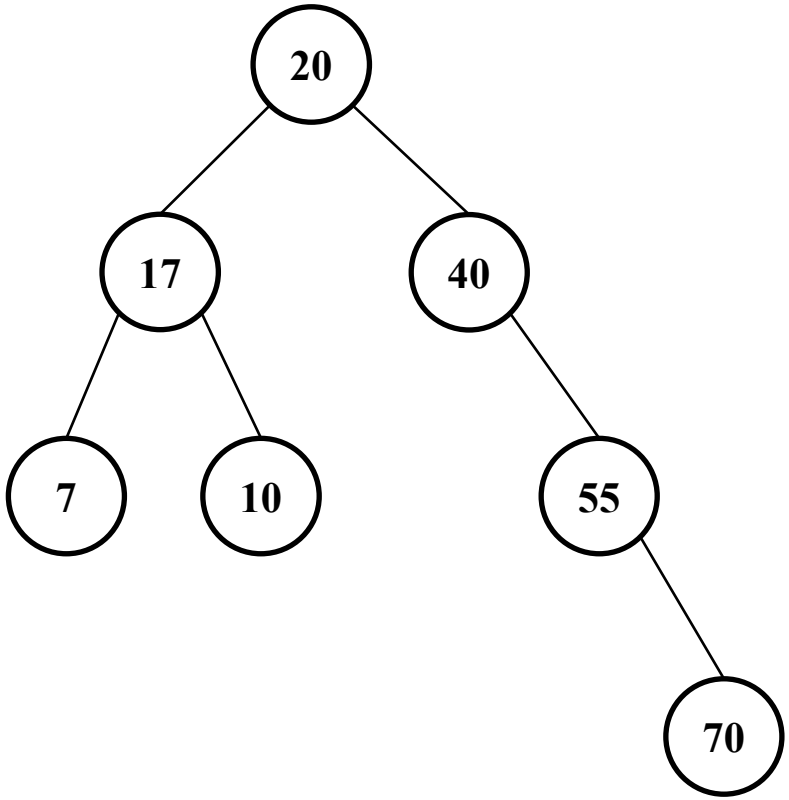
이진 탐색 트리 (2/3)

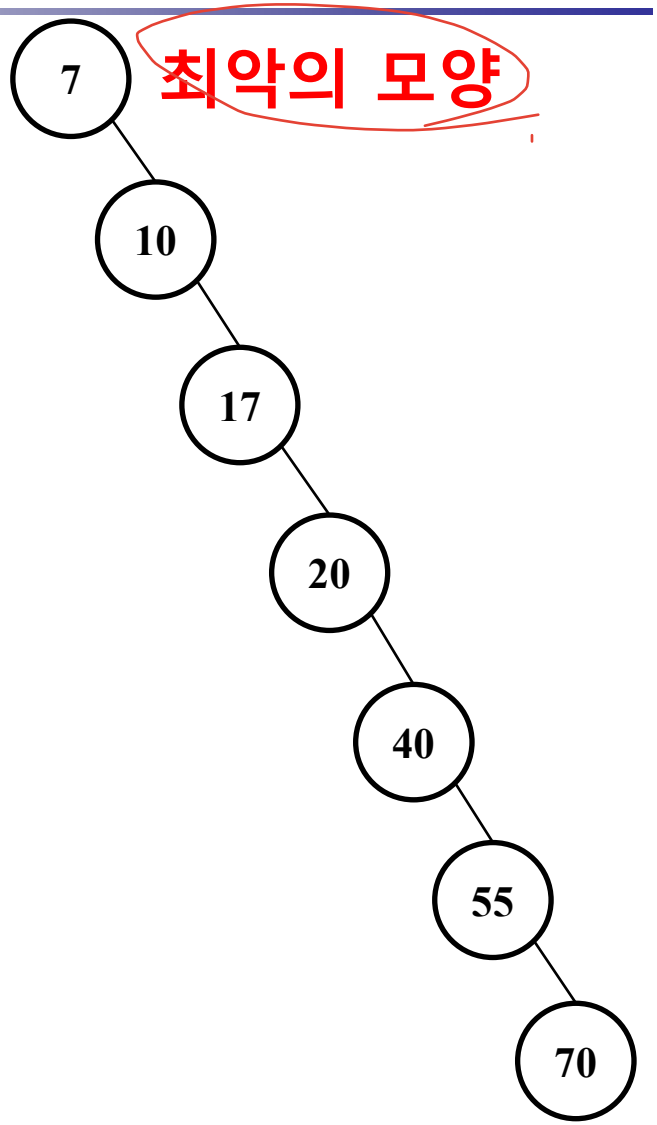
- 이진 탐색 트리
  - 같은 데이터와 다른 이진 탐색 트리 #1



이진 탐색 트리 (3/3)

- 이진 탐색 트리
  - 같은 데이터와 다른 이진 탐색 트리 #2





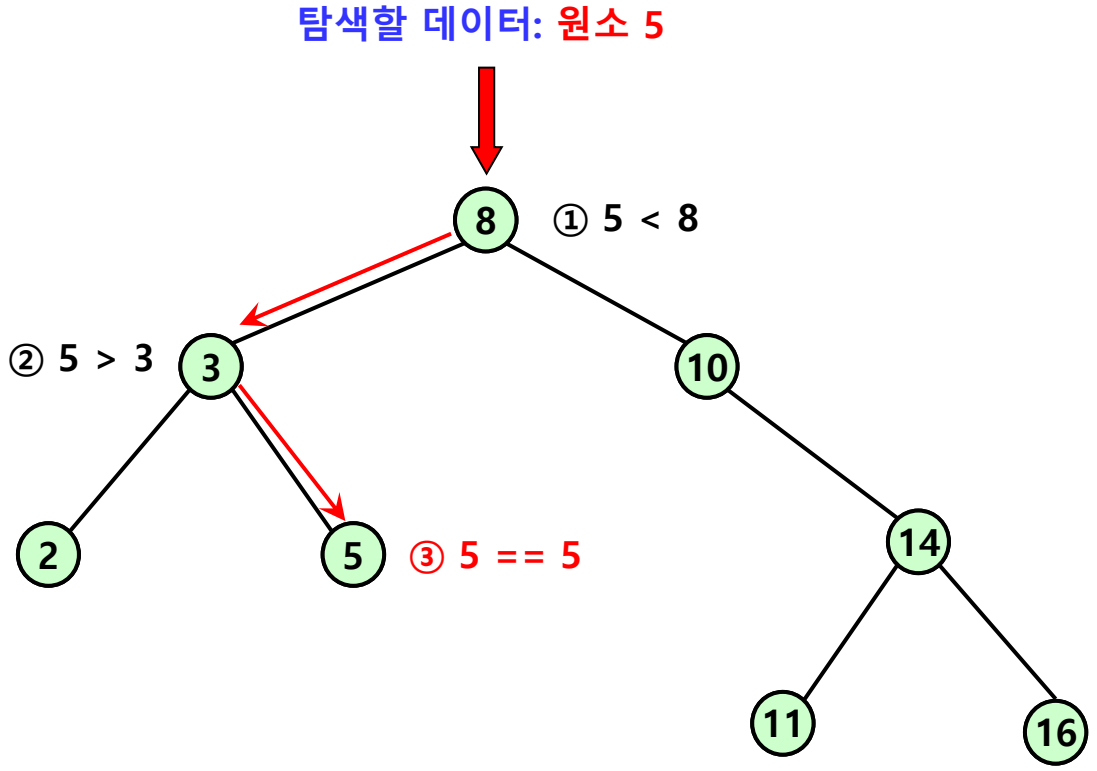
최악의 모양

이진 탐색 트리

이진 탐색 트리 연산

이진 탐색 트리 (1/7)

- **이진 탐색 트리:** 탐색
  - 탐색 과정

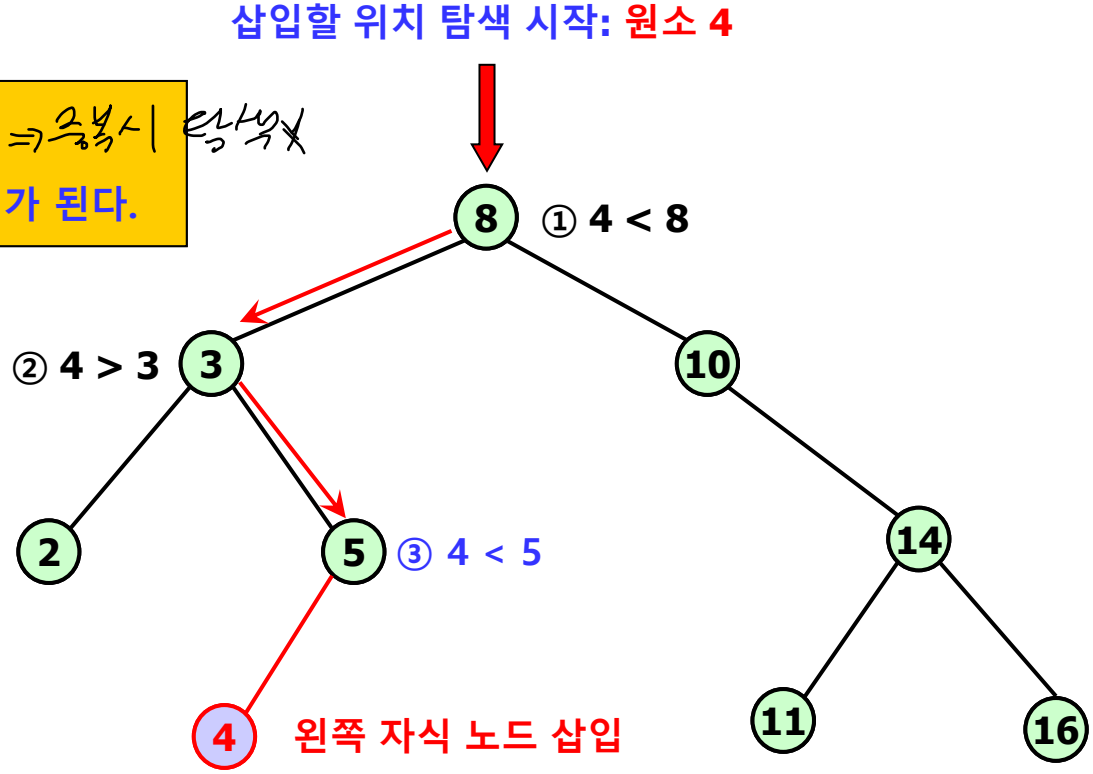


이진 탐색 트리 (2/7)

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

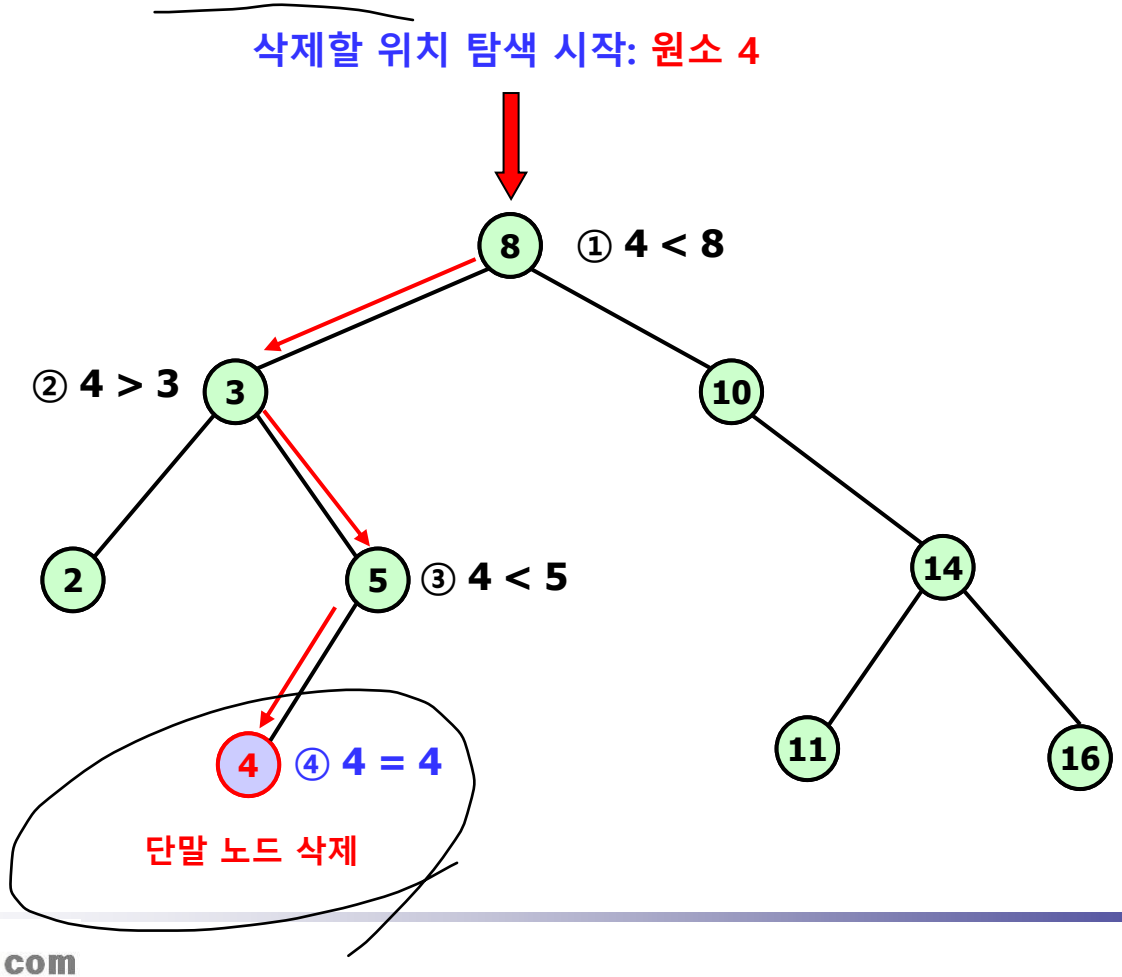
"탐색 실패가 결정 된 위치" ⇒ 즉, 탐색 X

즉, 왼쪽 자식 노드의 위치가 삽입 할 자리가 된다.



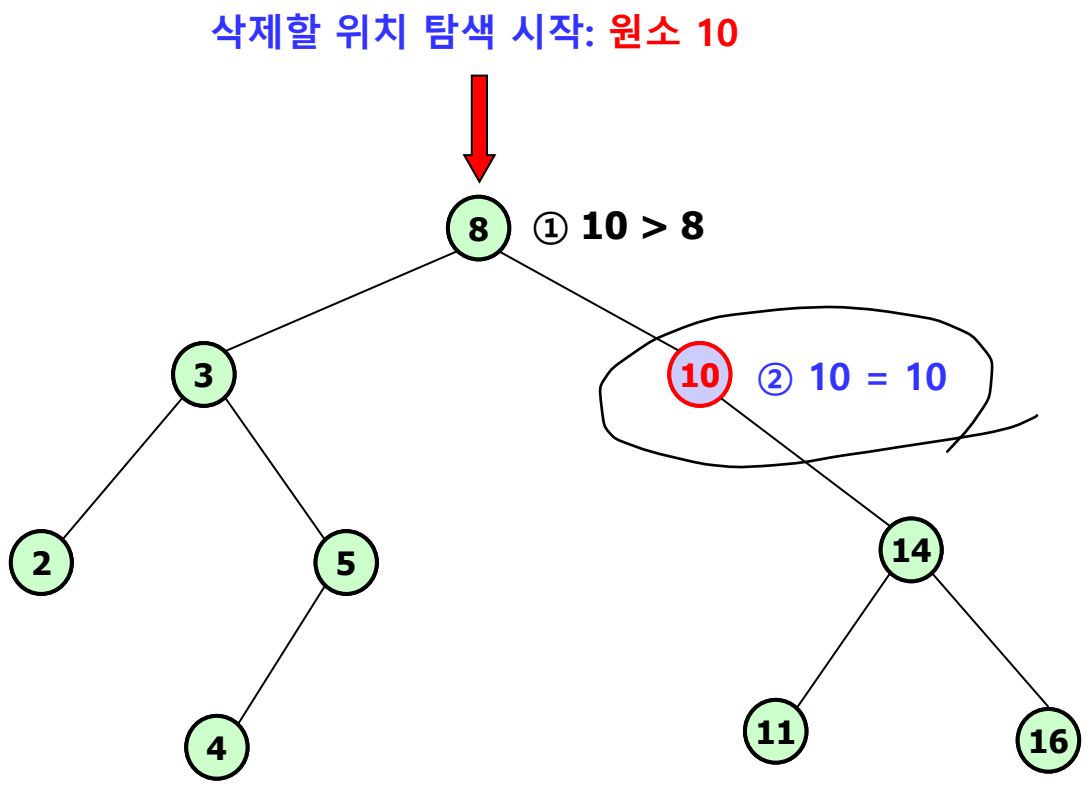
이진 탐색 트리 (3/7)

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



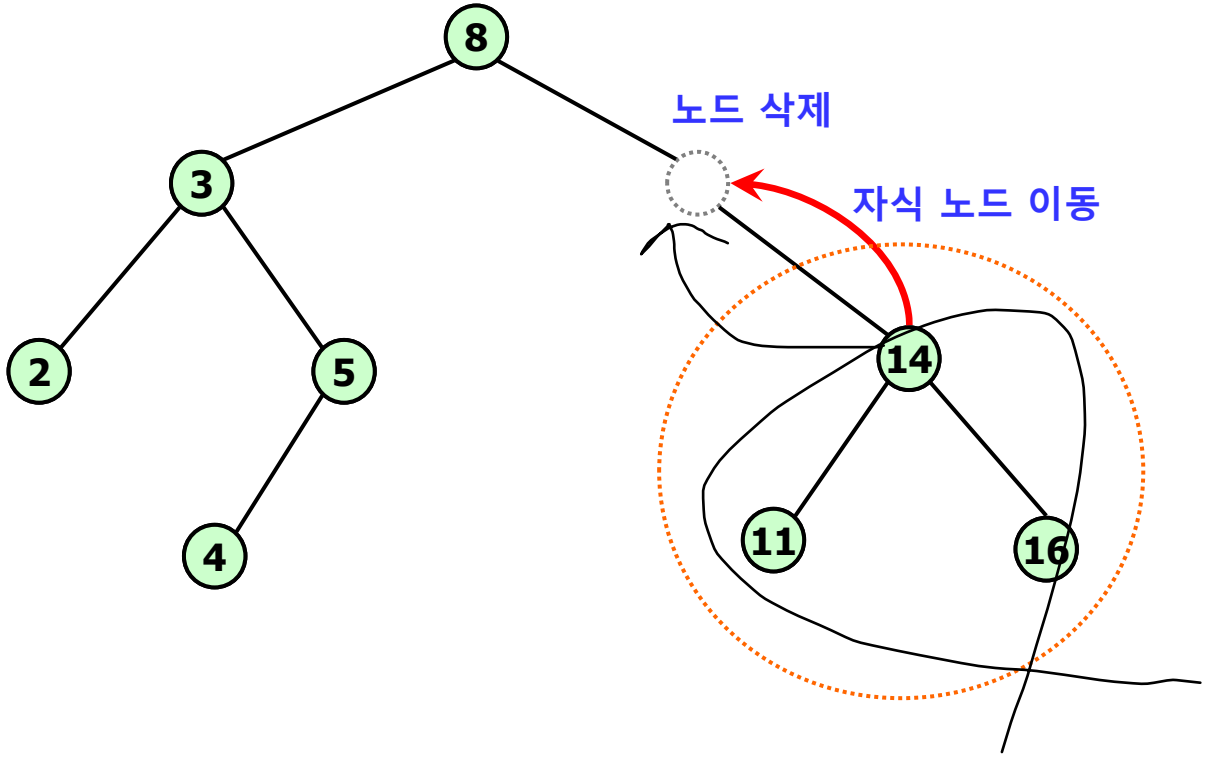
이진 탐색 트리 (4/7)

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



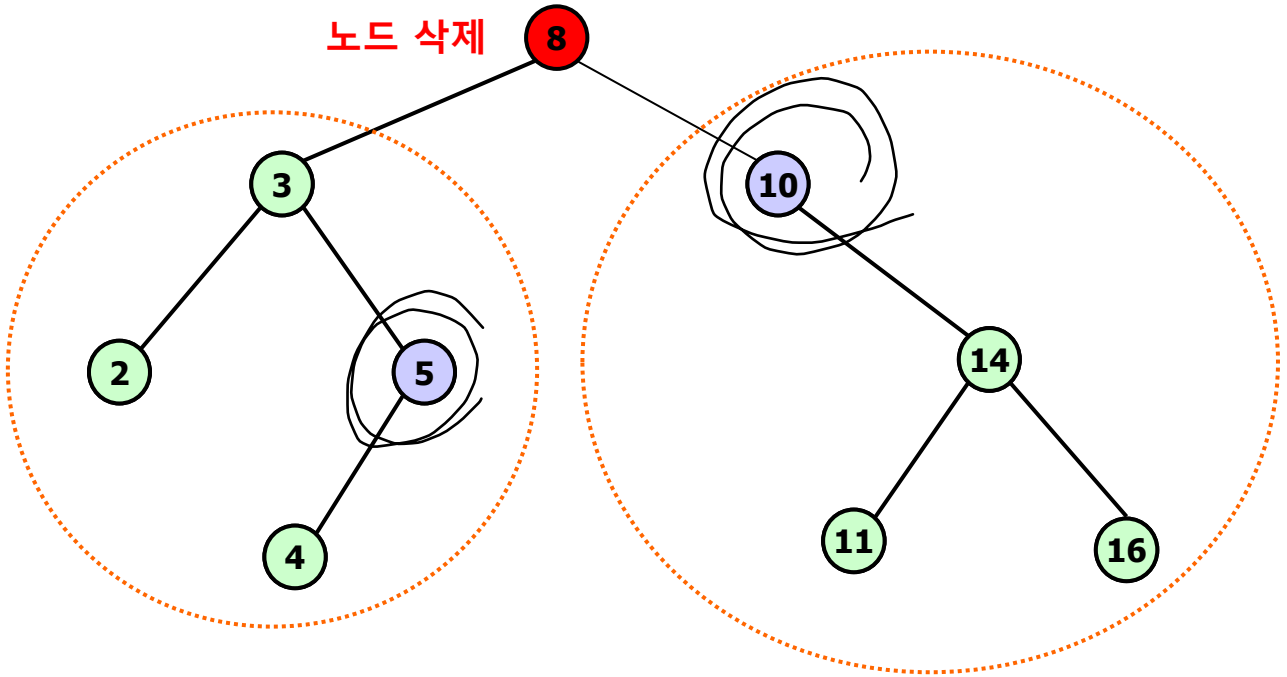
이진 탐색 트리 (5/7)

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



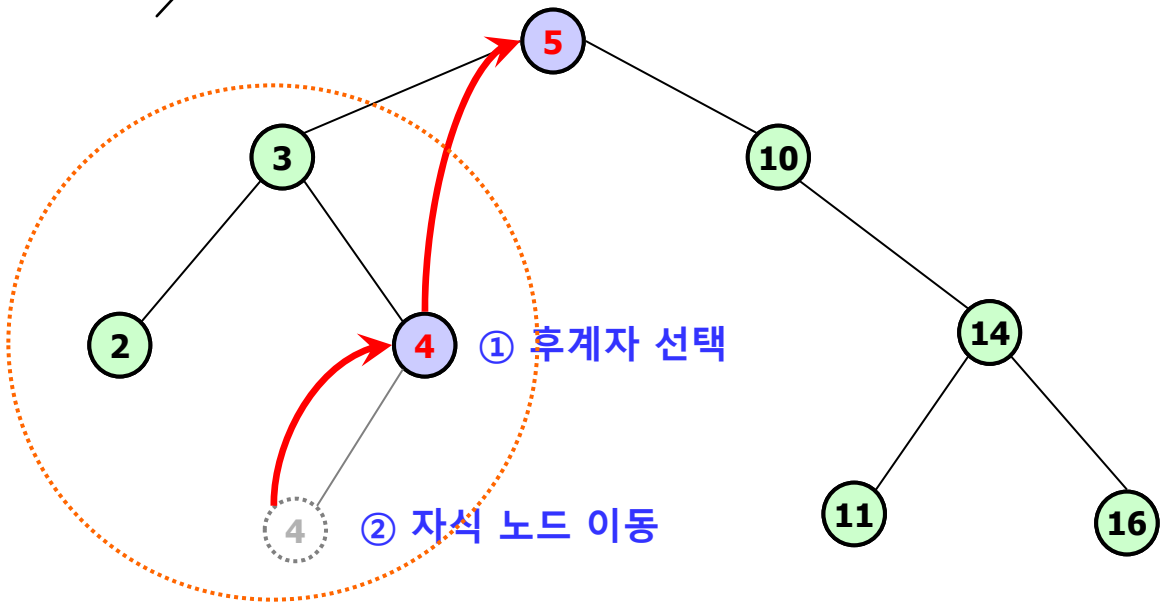
이진 탐색 트리 (6/7)

- **이진 탐색 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 탐색 트리 (7/7)

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



이진 탐색 트리

이진 탐색 트리 연산: 알고리즘

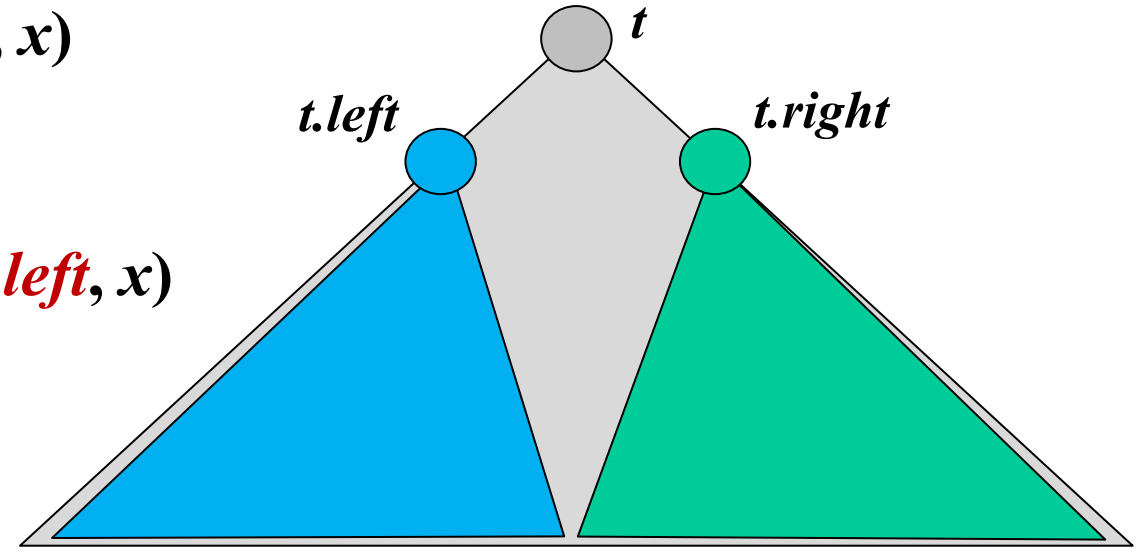
이진 탐색 트리 연산: 알고리즘 (1/3)

- 이진 탐색 트리 연산: 알고리즘(탐색) -- 재귀적 용법

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

search(t, x)

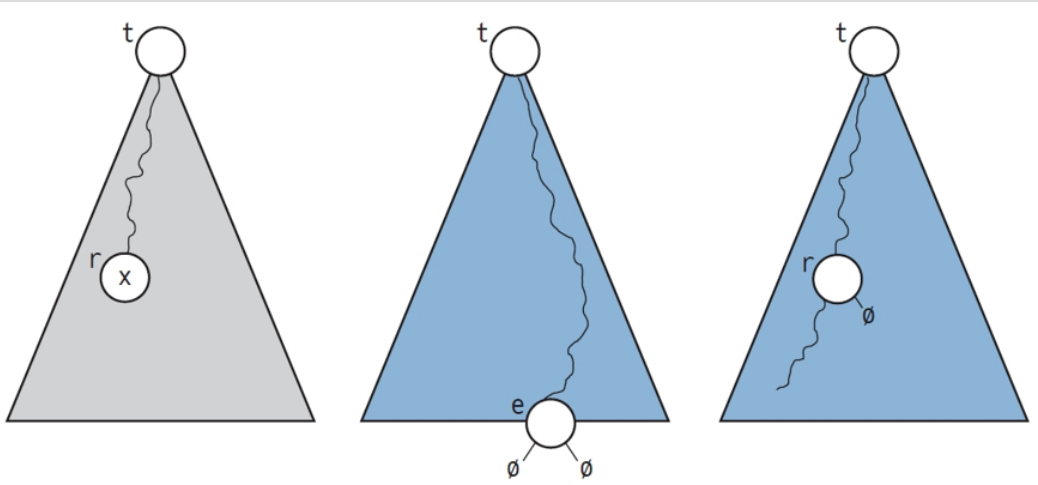
search($t.left$, x)



이진 탐색 트리 연산: 알고리즘 (2/3)

- 이진 탐색 트리 연산: 알고리즘(탐색) -- 비재귀적 용법

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

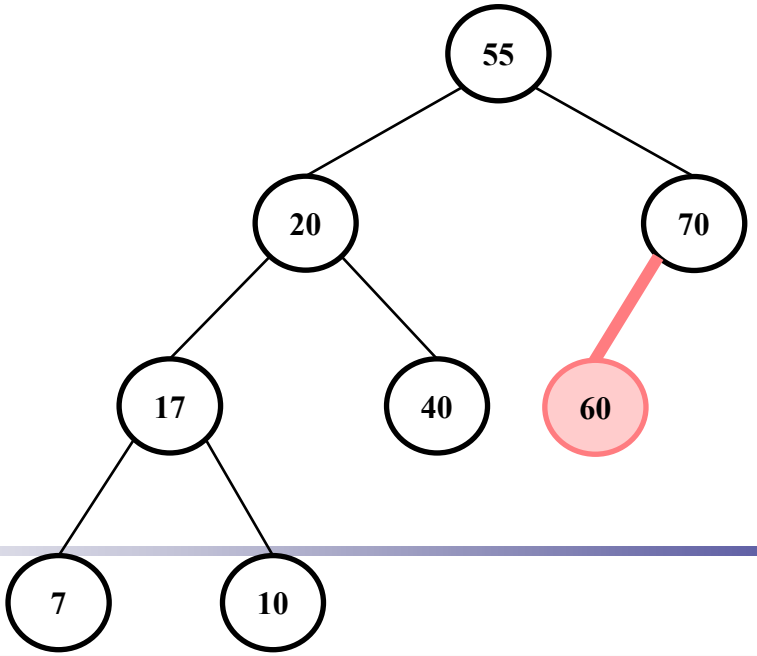


(a) 성공적인 검색 (b) 실패한 검색

이진 탐색 트리 연산: 알고리즘 (3/3)

- 이진 탐색 트리 연산: 알고리즘(삽입) -- 재귀적 용법

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



이진 탐색 트리 연산: 알고리즘 (4/3)

- 이진 탐색 트리 연산: 알고리즘(삽입) -- 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 (T = NULL) then                T ← newDNode;
    else if (data < parent.key) then  parent.Llink ← newDNode;
    else                              parent.Rlink ← newDNode;
end insertBST()
```

이진 탐색 트리 연산: 알고리즘 (5/3)

- 이진 탐색 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()
```

균형 탐색 트리

- 이진 탐색 트리
- **균형 탐색 트리**
  - AVL 트리
  - 레드-블랙 트리
  - B-트리

균형 탐색 트리

- **균형 탐색 트리(Binary Search Tree)**
  - 균형 탐색 트리

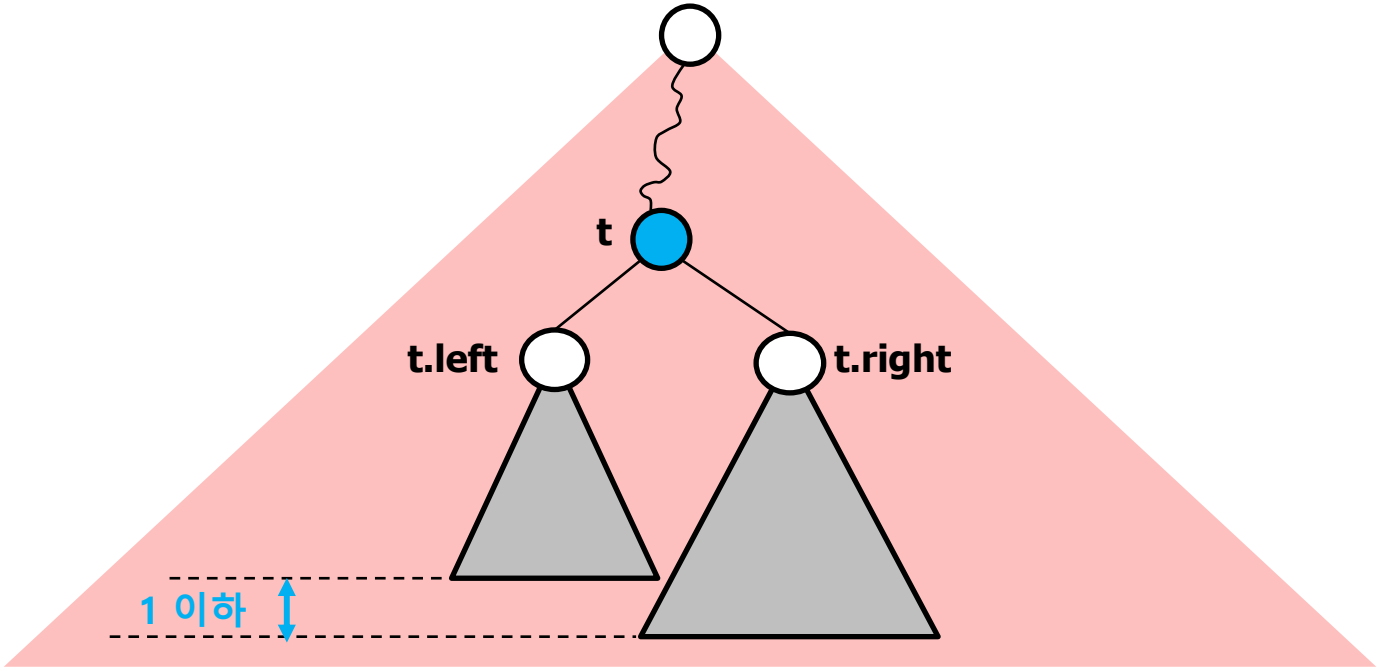
[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

균형 탐색 트리

AVL 트리

AVL 트리 (1/2)

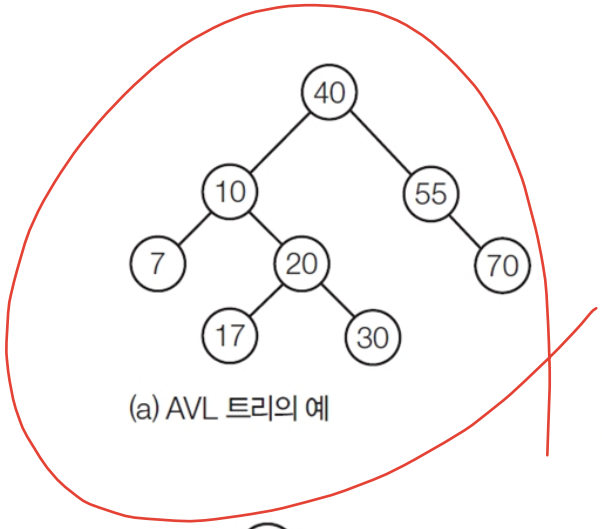
- **AVL 트리**
  - 모든 노드에 대해 좌 서브 트리의 높이(깊이)와 우 서브 트리의 높이의 차가 1을 넘지 않는다.



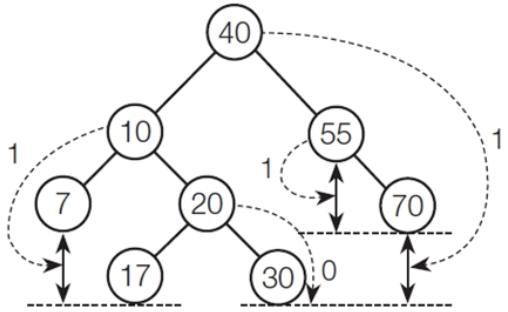
[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

AVL 트리 (2/2)

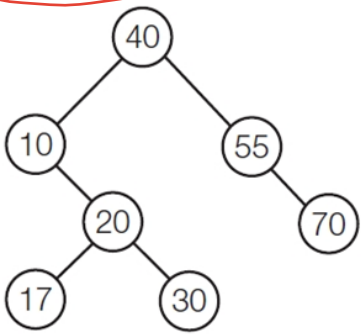
- **AVL 트리**
  - AVL 트리의 예



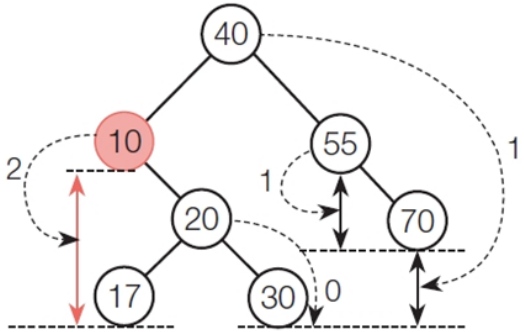
(a) AVL 트리의 예



(b) 서브 트리의 높이 차



(a) AVL 트리가 아닌 예



(b) AVL 트리가 아닌 이유: 높이 차 2

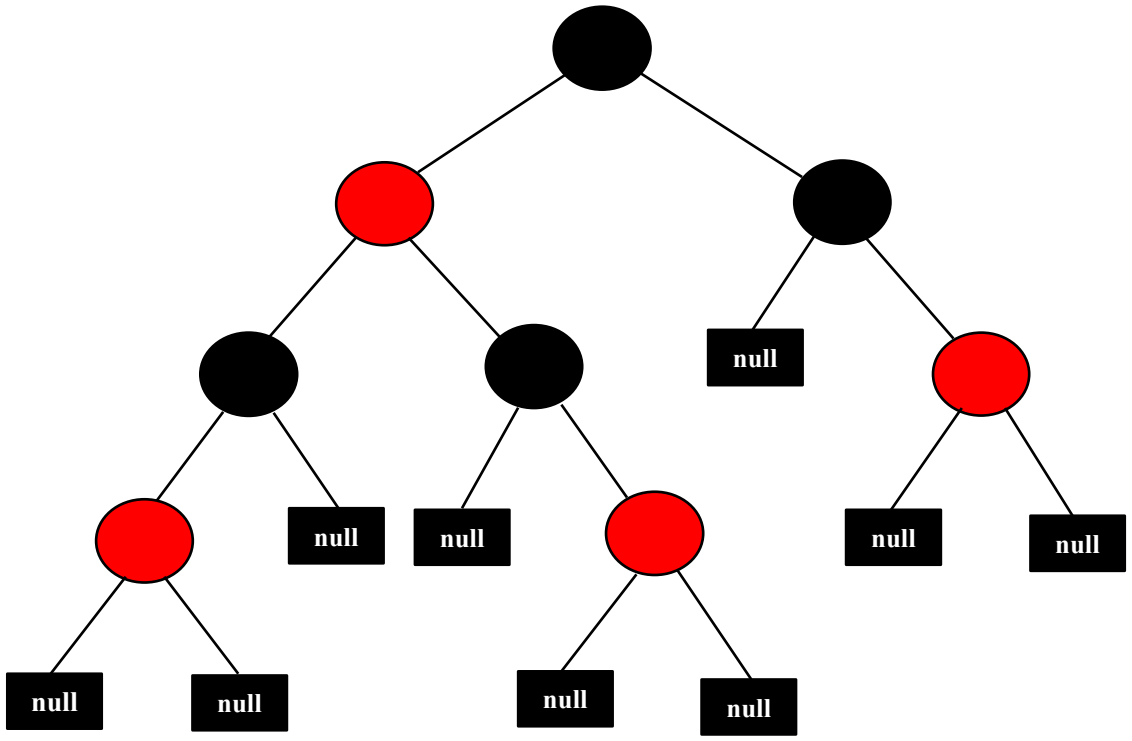
[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

균형 탐색 트리

레드-블랙 트리

레드-블랙 트리

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



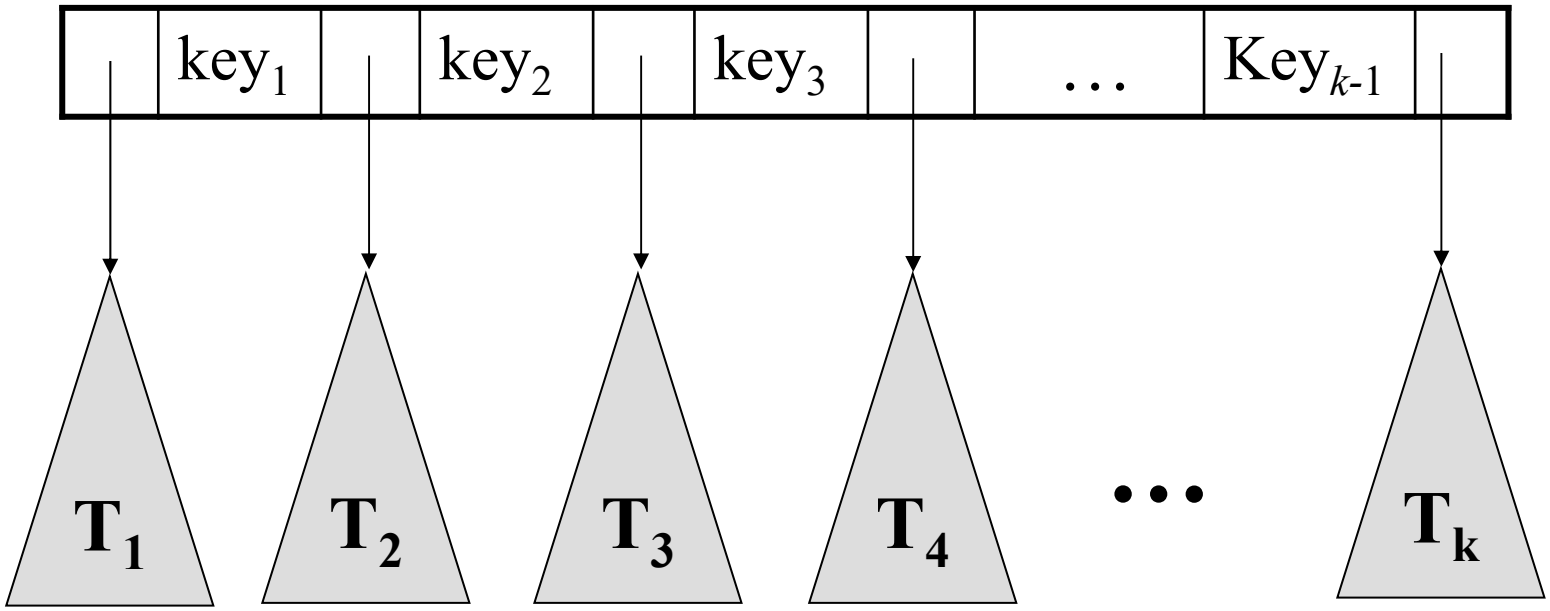
[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

균형 탐색 트리

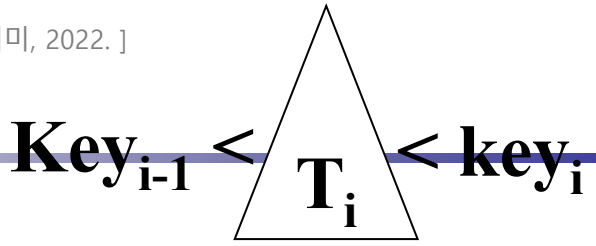
B 트리

B 트리

- **B 트리**
  - B-트리는 K-진 검색 트리가 균형을 유지하도록 하여 최악의 경우 디스크 접근 횟수를 줄인 것이다.



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



참고문헌

[1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, “C++로 구현하는 자료구조와 알고리즘”, 한티에듀, 2020.

[2] 주우석, “IT CookBook, C・C++ 로 배우는 자료구조론”, 한빛아카데미, 2019.

[3] 이지영, “C 로 배우는 쉬운 자료구조”, 한빛아카데미, 2022.

[4] “IT CookBook, 쉽게 배우는 자료구조 with 파이썬”, 문병로, 한빛아카데미, 2022.

[5] “프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문”, 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.

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

[7] 문병로, “IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법”(개정판), 개정판, 한빛아카데미, 2018.

[8] Richard E. Neapolitan, 도경구 역, “알고리즘 기초”, 도서출판 홍릉, 2017.