트리 3

–탐색과 이진 탐색 트리–

HaRim Jung, Ph.D.

Visiting Professor / Senior Researcher

SKKU Institute for Convergence / Convergence Research Institute

Sungkyunkwan University, Korea

탐색

❑ 탐색(Search)의 개요

- **탐색**

  따라서 탐색의 시간 효율성을 증가시킬 수 있는 자료구조 및 알고리즘은 필수적

  – 컴퓨터가 **가장 많이 수행하는 연산** 중 하나로 다수의 항목 중 원하는 항목을 찾는 연산

- **기본적인 탐색 알고리즘**

  – **선형 탐색(Linear Search)**: 정렬되지 않은 Python 리스트 혹은 연결 리스트에서 사용하는 알고리즘으로, 처음 항목부터 마지막 항목까지 하나씩 검사하면서 특정 값(검색 키: Search Key)과 동일한 키 값을 가지는 항목을 찾는 방법

    • 시간 복잡도: O(N)

  – **이진 탐색(Binary Search)**: **정렬된** Python 리스트(혹은 정적·동적 배열)를 중간에 위치한 항목을 기준으로 두 부분으로 나누어 가며 검색 키와 동일한 키 값을 가지는 항목을 찾는 방법

    • 시간 복잡도: O(logN)

```python
def linear_search(lst, n, x):
    for i in range(n):
        if lst[i] == x:
            return i
    return -1

lst = [10, 7, 11, 5, 3, 8, 16, 13]
n = len(lst)
result = linear_search(lst, n, 30)
if result != -1:
    print("Found at index {}.".format(result))
else:
    print("Not found.")
```

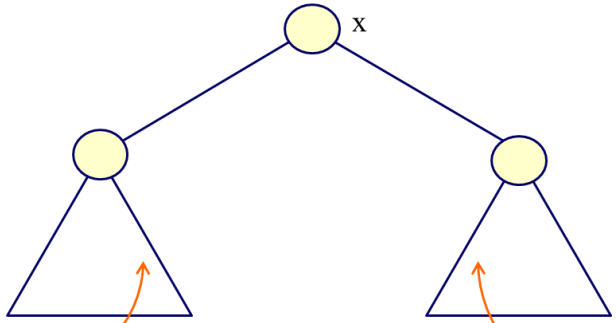
```python
def binary_search(lst, n, x):
    mid = 0
    low = 0
    high = n - 1
    while low <= high:
        mid = (low + high) // 2
        if x == lst[mid]:
            return mid
        if x < lst[mid]:
            high = mid - 1
        else:
            low = mid + 1
    return -1

lst = [3, 5, 7, 8, 10, 11, 13, 16]
n = len(lst)
result = binary_search(lst, n, 30)
if result != -1:
    print("Found at index {}.".format(result))
else:
    print("Not found.")
```

이진 탐색 트리 (1/22)

❑ 이진 탐색 트리(BST: Binary Search Tree)

따라서 이진 탐색 트리를 중위 순회하면서 노드의 키 값을 출력하면, 키 값들이 정렬되어 출력

- **이진 탐색의 개념을 트리 구조에 접목한 자료구조**
  - **(정의)** 이진 탐색 트리는 다음과 같은 조건을 만족하는 이진 트리임
    - 모든 노드들의 키는 서로 다른 **유일한 값**을 가짐
    - 특정 노드 N의 키 값이 N의 왼쪽 서브 트리에 존재하는 모든 노드들의 키 값보다 크고, N의 오른쪽 서브 트리에 있는 모든 노드들의 키 값보다 작음

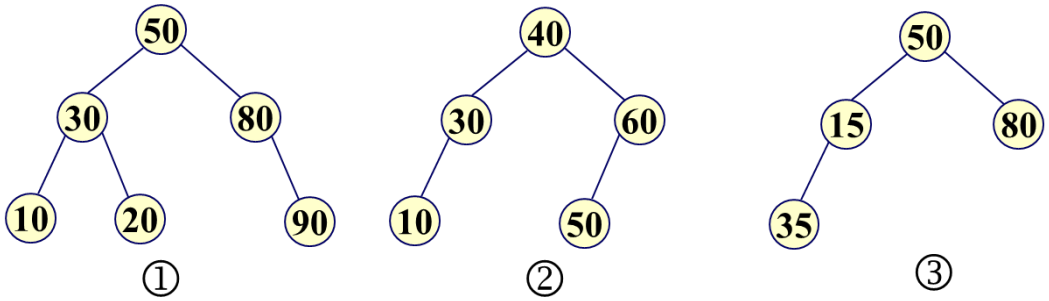


for any node y in this subtree, key(y) < key(x)

for any node z in this subtree, key(z) > key(x)

- N의 왼쪽 서브 트리와 오른쪽 서브 트리도 **이진 탐색 트리**임

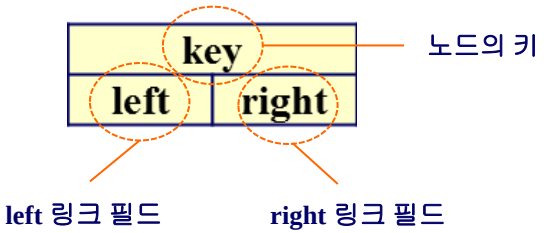
다음 중 이진 탐색 트리는?



이진 탐색 트리 (2/22)

❑ 이진 탐색 트리의 구현

- 노드들을 참조를 이용하여 연결 시켜 구현
- 노드의 구조
  - 이진 트리의 노드 구조와 동일




```
1  class Node:
2      def __init__(self, item, left_child = None, right_child = None):
3          self.key = item
4          self.left = left_child
5          self.right = right_child
6      def get_key(self): return self.key
7      def get_left(self): return self.left
8      def get_right(self): return self.right
9      def set_key(self, new_item):
10         self.key = new_item
11     def set_left(self, new_left_child):
12         self.left = new_left_child
13     def set_right(self, new_right_child):
14         self.right = new_right_child
```

노드 객체를 위한 Node 클래스

btnode.py

- 이진 탐색 트리에 적용 가능한 주요 연산
  - **BST()**: 빈 이진 탐색 트리 생성
  - **search(k)**: 검색 키(search key) 값인 k와 같은 키 값을 가지는 노드가 존재하면 True 반환
  - **insert(key)**: 키 값 key를 가지는 노드 삽입
  - **find_min()**: 키 값이 최소인 노드 탐색
  - **delete_min()**: 키 값이 최소인 노드 삭제
  - **delete(key)**: 키 값 key를 가지는 노드 삭제

⋮

이진 탐색 트리 (3/22)

❑ 이진 탐색 트리의 구현 contd.

- 이진 탐색 트리 객체를 위한 클래스 정의

```
from btnode import Node
class BST:
    def __init__(self):
        self.root = None

    def search(self, k):
        return self._search(self.root, k)
    def _search(self, n, k):
        if n == None or n.get_key() == k:
            return n != None
        elif n.get_key() > k:
            return self._search(n.get_left(), k)
        else:
            return self._search(n.get_right(), k)

    def insert(self, key):
        self.root = self._insert(self.root, key)
    def _insert(self, n, key):
        if n == None:
            return Node(key)
        if n.get_key() > key:
            n.set_left(self._insert(n.get_left(), key))
        elif n.get_key() < key:
            n.set_right(self._insert(n.get_right(), key))
        return n

    def find_min(self):
        if self.root == None:
            return None
        return self._find_min(self.root)
    def _find_min(self, n):
        if n.get_left() == None:
            return n
        return self. find min(n.get left())
```

계속

```
def delete_min(self):
    if self.root == None:
        print("Tree is empty.")
    self.root = self._delete_min(self.root)
def _delete_min(self, n):
    if n.get_left() == None:
        return n.get_right()
    n.set_left(self._delete_min(n.get_left()))
    return n

def delete(self, key):
    self.root = self._delete(self.root, key)
def _delete(self, n, key):
    if n == None:
        return None
    if n.get_key() > key:
        n.set_left(self._delete(n.get_left(), key))
    elif n.get_key() < key:
        n.set_right(self._delete(n.get_right(), key))
    else:
        if n.get_left() == None and n.get_right() == None:
            return None
        if n.get_left() == None or n.get_right() == None:
            if n.get_left() == None:
                return n.get_right()
            else:
                return n.get_left()
        target = n
        n = self._find_min(target.get_right())
        n.set_right(self._delete_min(target.get_right()))
        n.set_left(target.get_left())
    return n
```

이진 탐색 트리 (4/22)

❑ 이진 탐색 트리의 구현 contd.

- **BST()** 시간복잡도: O(1)

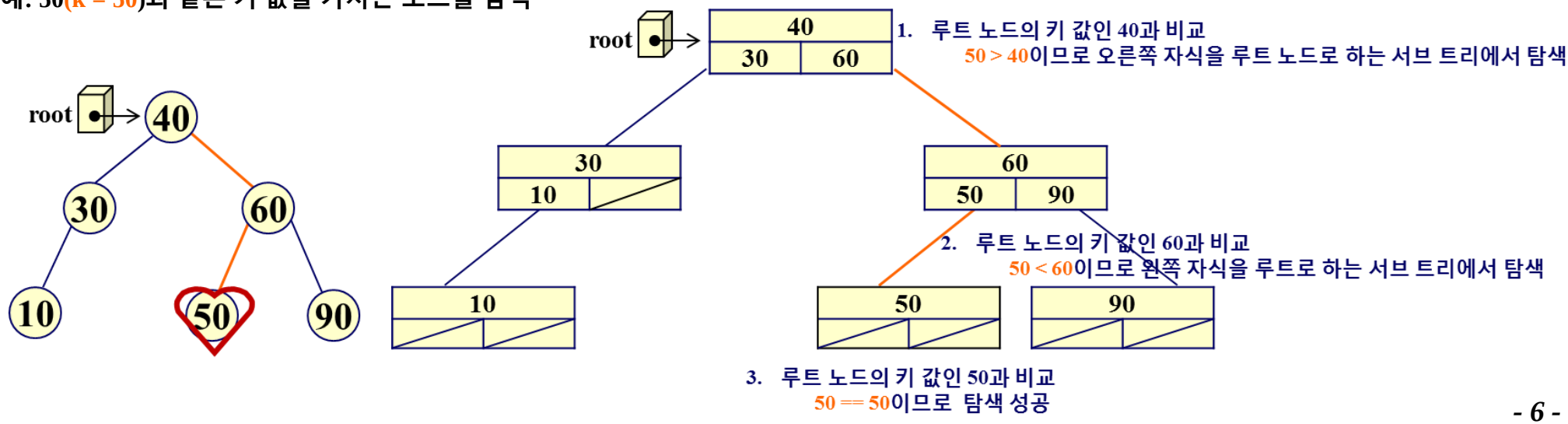
```
3      def __init__(self):
4          self.root = None
```

  - 빈(empty) 이진 탐색 트리를 생성
    - **라인 4**: 빈 이진 탐색 트리이므로 트리의 루트 노드의 참조를 저장하는 변수인 root를 None으로 설정

-이진 탐색 트리 탐색 연산-

1. 검색키 값이 k라면, 루트 노드의 키 값과 k를 비교하는 것으로 탐색을 시작
2. (i) k가 루트 노드의 키 값보다 작으면, 루트 노드의 왼쪽 서브 트리에서 k와 같은 키 값을 가지는 노드를 찾고, (ii) 크면 루트 노드의 오른쪽 서브 트리에서 k와 같은 키 값을 가지는 노드를 찾으며, (iii) 같으면 탐색 성공
3. 왼쪽 ·오른쪽 서브 트리에서의 탐색은 재귀적으로 반복

예: 50(k = 50)과 같은 키 값을 가지는 노드를 탐색



이진 탐색 트리 (5/22)

❑ 이진 탐색 트리의 구현 contd.

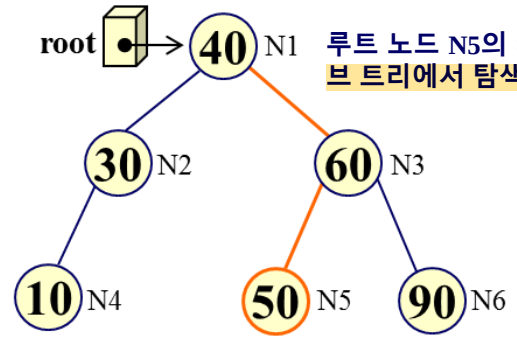
- **search() 시간복잡도: O(h), 여기서 h는 트리의 높이(일반적으로 O(logN), 편향된 이진 탐색 트리의 경우 O(N))**

```
6     def search(self, k):
7         return self._search(self.root, k)
8     def _search(self, n, k):
9         if n == None or n.get_key() == k:
10            return n != None
11        elif n.get_key() > k:
12            return self._search(n.get_left(), k)
13        else:
14            return self._search(n.get_right(), k)
```

  - 검색 키(search key) 값인 k와 같은 키 값을 가지는 노드가 존재하면 True 반환
    - **라인 7:** protected 멤버함수(메소드) _search() 호출 (인자는 **루트 노드**와 검색 키 값 **k**)
  - **_search(self, n, k)**
    - **라인 9-10: if** 루트 노드가 None이면 False를 반환하고 루트 노드의 키 값이 k와 같다면 True를 반환(종료 조건)

k = 45

해당 트리는 None → False를 반환 root



루트 노드 N5의 키 값인 50보다 k가 작으므로 N5의 왼쪽 자식을 루트 노드로 하는 서브 트리에서 탐색해야 하지만 해당 트리는 None → False를 반환

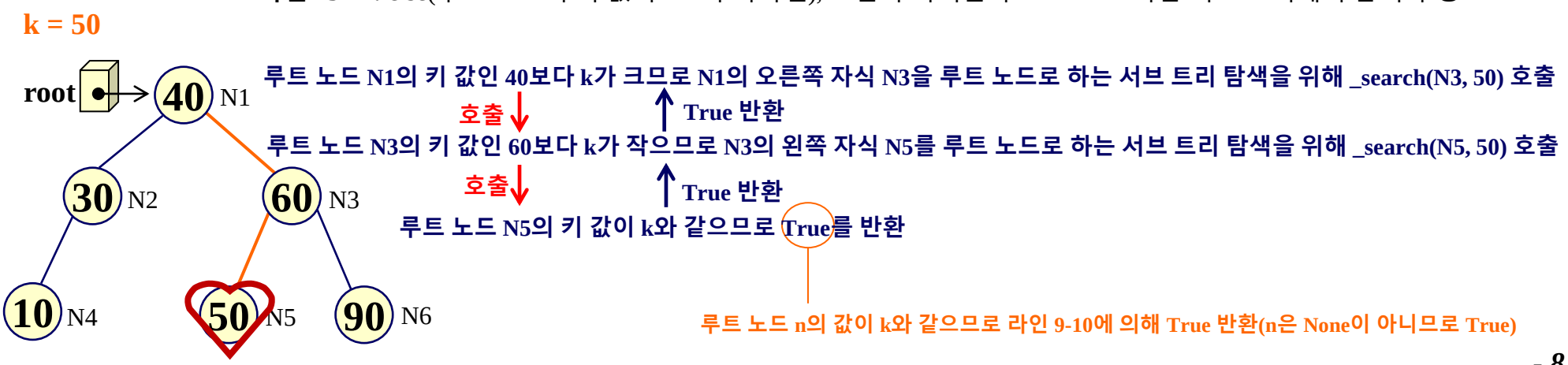
이진 탐색 트리 (6/22)

❑ 이진 탐색 트리의 구현 contd.

- **search() contd.**

```
6     def search(self, k):
7         return self._search(self.root, k)
8     def _search(self, n, k):
9         if n == None or n.get_key() == k:
10            return n != None
11        elif n.get_key() > k:
12            return self._search(n.get_left(), k)
13        else:
14            return self._search(n.get_right(), k)
```

  - **_search(self, n, k) contd.**
    - **라인 11-12: elif** 루트 노드의 키 값이 k보다 크다면, 왼쪽 자식을 루트 노드로 하는 서브 트리에서 탐색 수행
    - **라인 13-14: else**(루트 노드의 키 값이 k보다 작다면), 오른쪽 자식을 루트 노드로 하는 서브 트리에서 탐색 수행



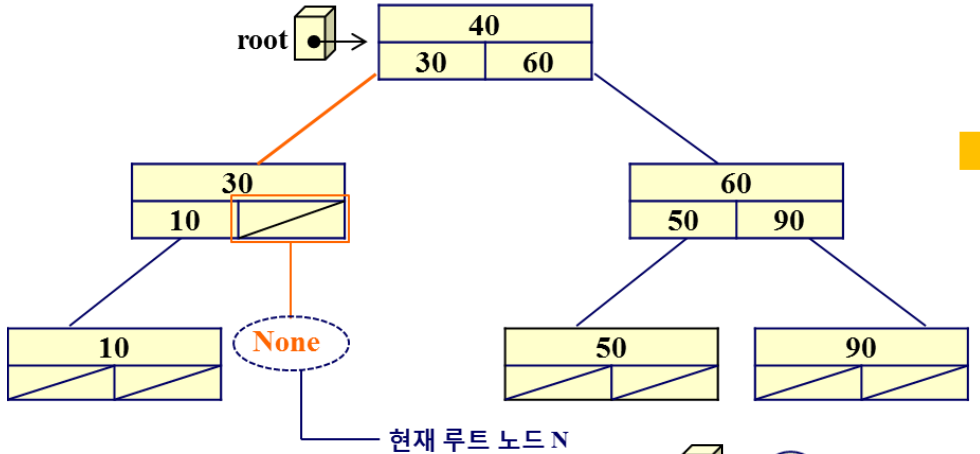
이진 탐색 트리 (7/22)

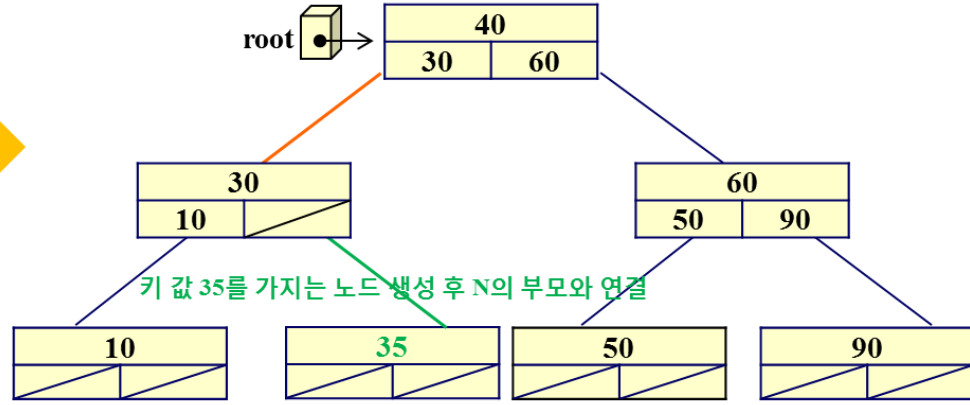
❑ 이진 탐색 트리의 구현 contd.

-이진 탐색 트리 삽입 연산-

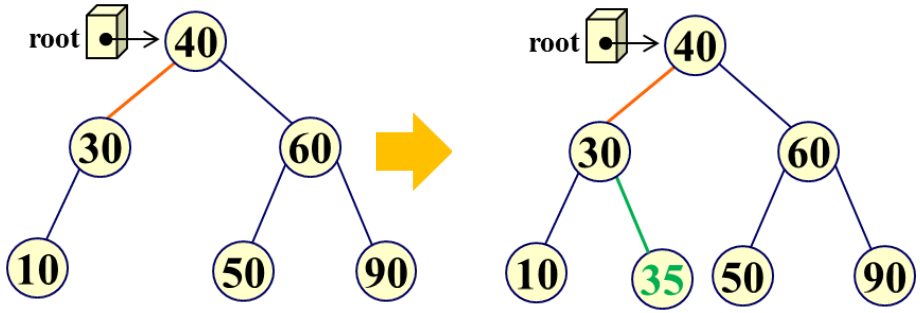
삽입 연산은 탐색 연산과 거의 동일

1. 탐색 연산을 수행
2. 탐색 중 루트 노드 N == None인 서브 트리를 만나면 새 노드를 생성하여 N의 부모 노드와 연결(삽입)
   - 단, 이미 동일한 키 값을 가지는 노드가 존재하는 경우 삽입하지 않음 왜? 이진 탐색 트리에서 모든 노드의 키는 서로 다른 유일한 값을 가지므로





예: 35를 키 값으로 가지는 노드를 삽입



[연습] 다음과 같은 키 값을 가지는 노드를 순서대로 삽입하여 생성한 이진 탐색 트리를 그리시오..

17, 10, 22, 15, 13, 24, 20, 11, 14	10, 9, 8, 7, 6, 5, 4, 3, 2, 1

이진 탐색 트리 (8/22)

❑ 이진 탐색 트리의 구현 contd.

- **insert(k) 시간복잡도: O(h), 여기서 h는 트리의 높이(일반적으로 O(logN), 편향된 이진 탐색 트리의 경우 O(N))**

```
16      def insert(self, key):
17          self.root = self._insert(self.root, key)
18      def _insert(self, n, key):
19          if n == None:
20              return Node(key)
21          if n.get_key() > key:
22              n.set_left(self._insert(n.get_left(), key))
23          elif n.get_key() < key:
24              n.set_right(self._insert(n.get_right(), key))
25          return n
```

  - 키 값 key를 가지는 노드 삽입
    - **라인 7:** protected 멤버함수 _insert() 호출 (인자는 **루트 노드**와 키 값 **key**)
  - **_insert(self, n, key)**
    - **라인 19-20**: **if** 루트 노드 n이 None이면(**조건 1**), key를 키 값으로 가지는 새로운 노드 Nnew 생성 후 Nnew 반환
    - **라인 21-22: if** 루트 노드 n의 키 값이 key보다 크다면(**조건 2**), 왼쪽 자식을 루트 노드로 하는 서브 트리에서 삽입 수행
    - **라인 23-24: elif** 루트 노드 n의 키 값이 key보다 작다면(**조건 3**), 오른쪽 자식을 루트 노드로 하는 서브 트리에서 삽입 수행

      Note: else를 사용하면 루트 노드의 키 값이 key와 같아도 재귀 호출이 이루어지므로 최종적으로 라인 19-20을 수행하게 됨. 따라서 반드시 라인 23처럼 작성해야 함(이진 탐색 트리는 동일한 키 값을 가지는 노드가 중복으로 존재하면 안됨)

    - **라인 25: 부모 노드와 연결하기 위해** (현재 서브 트리의) 루트 노드 n 반환

이진 탐색 트리 (9/22)

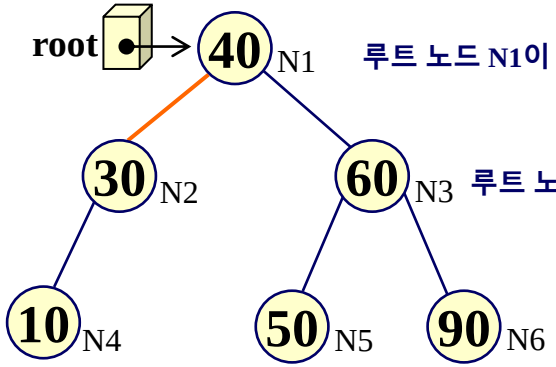
❑ 이진 탐색 트리의 구현 contd.

- **insert(k) contd.**

```
16      def insert(self, key):
17          self.root = self._insert(self.root, key)
18      def _insert(self, n, key):
19  조건1   if n == None:
20              return Node(key)
21  조건2   if n.get_key() > key:
22              n.set_left(self._insert(n.get_left(), key))
23  조건3   elif n.get_key() < key:
24              n.set_right(self._insert(n.get_right(), key))
25          return n
```

예: 중복된 키 값을 가지는 노드 삽입 시도 시 라인 23이 elif n.get_key() < key:인 경우

key = 30



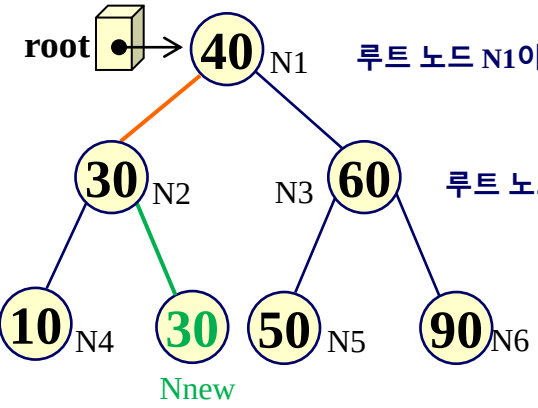
루트 노드 N1이 조건 2를 만족하므로 N1의 왼쪽 자식 N2을 루트 노드로 하는 서브 트리에서 삽입하기 위해 _insert(N2, 30) 호출

호출 ↓ ↑ N2 반환

루트 노드 N2가 조건 1, 2, 3을 만족하지 못하므로 현재 루트 노드인 본인 N2 반환 (라인 25)

예: 중복된 키 값을 가지는 노드 삽입 시도 시 라인 23이 else:인 경우 중복된 키 값을 가진 노드가 삽입 되므로 잘못된 이진 탐색 트리가 된다

key = 30



루트 노드 N1이 조건 2를 만족하므로 N1의 왼쪽 자식 N2을 루트 노드로 하는 서브 트리에서 삽입하기 위해 _insert(N2, 30) 호출

호출 ↓ ↑ N2 반환

루트 노드 N2가 조건 3을 만족하므로 N2의 오른쪽 자식 None을 루트 노드로 하는 서브 트리에서 삽입하기 위해 _insert(None, 30) 호출

호출 ↓ ↑ Nnew 반환

루트 노드 None이 조건 1을 만족하므로 새로운 노드 Nnew를 생성 후 None의 부모 노드 N2에게 반환

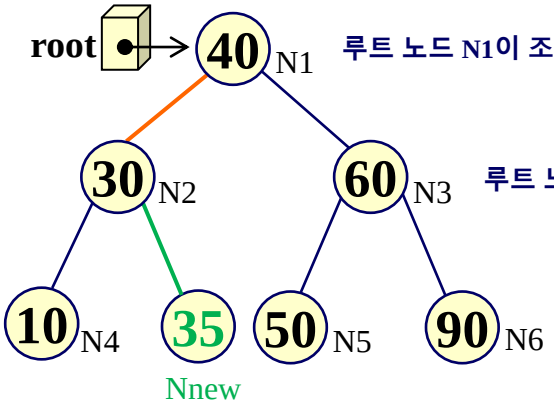
이진 탐색 트리 (10/22)

❑ 이진 탐색 트리의 구현 contd.

- **insert(k) contd.**

```
16      def insert(self, key):
17          self.root = self._insert(self.root, key)
18      def _insert(self, n, key):
19  조건1   if n == None:
20              return Node(key)
21  조건2   if n.get_key() > key:
22              n.set_left(self._insert(n.get_left(), key))
23  조건3   elif n.get_key() < key:
24              n.set_right(self._insert(n.get_right(), key))
25          return n
```

예: 35를 키 값으로 가지는 노드를 삽입



루트 노드 N1이 조건 2를 만족하므로 N1의 왼쪽 자식 N2을 루트 노드로 하는 서브 트리에서 삽입하기 위해 _insert(N2, 35) 호출

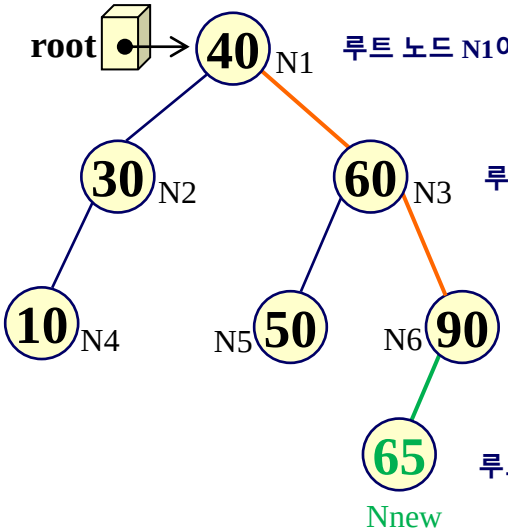
호출 ↓ ↑ N2 반환

루트 노드 N2가 조건 3을 만족하므로 N2의 오른쪽 자식 None을 루트 노드로 하는 서브 트리에서 삽입하기 위해 _insert(None, 35) 호출

호출 ↓ ↑ Nnew 반환

루트 노드 None이 조건 1을 만족하므로 35를 키 값으로 가지는 새로운 노드 Nnew를 생성 후 None의 부모 노드 N2에게 반환

예: 65를 키 값으로 가지는 노드를 삽입



루트 노드 N1이 조건 3을 만족하므로 N1의 오른쪽 자식 N3을 루트 노드로 하는 서브 트리에서 삽입하기 위해 _insert(N3, 65) 호출

호출 ↓ ↑ N3 반환

루트 노드 N3이 조건 3을 만족하므로 N3의 오른쪽 자식 N6을 루트 노드로 하는 서브 트리에서 삽입하기 위해 _insert(N6, 65) 호출

호출 ↓ ↑ N6 반환

루트 노드 N6이 조건 2를 만족하므로 N6의 왼쪽 자식 None을 루트 노드로 하는 서브 트리에서 삽입하기 위해 _insert(None, 65) 호출

호출 ↓ ↑ Nnew 반환

루트 노드 None이 조건 1을 만족하므로 새로운 노드 Nnew를 생성 후 None의 부모 노드 N6에게 반환

이진 탐색 트리 (11/22)

❑ 이진 탐색 트리의 구현 contd.

-이진 탐색 트리에서 키 값이 최소인 노드 탐색 및 키 값이 최소인 노드 삭제-

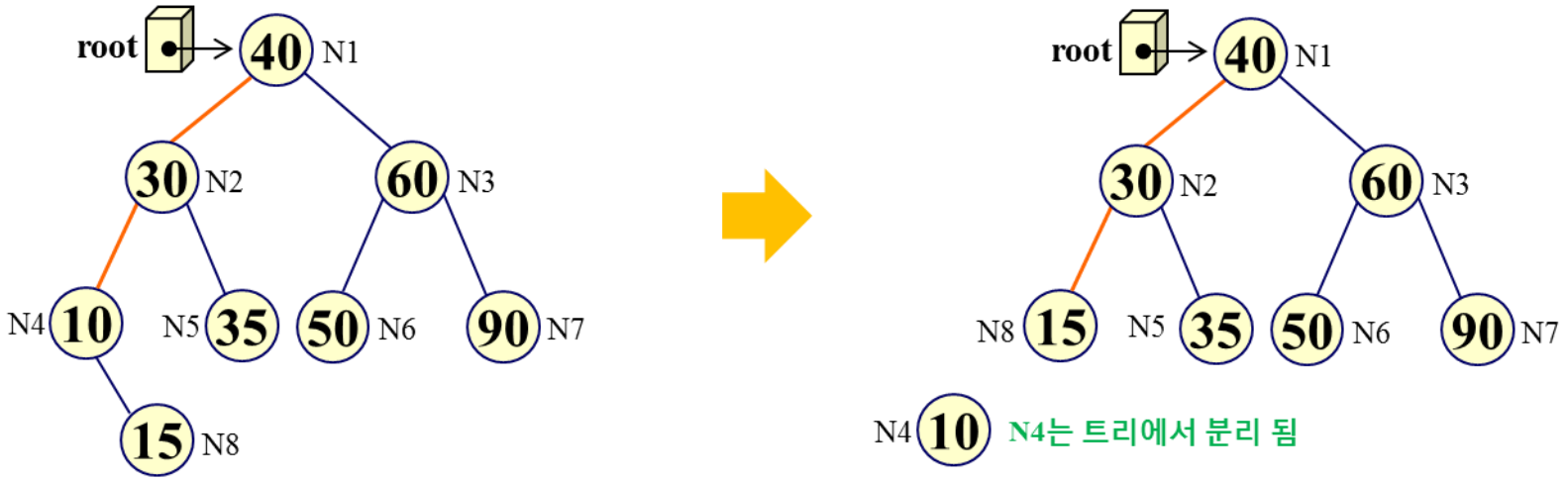
이진 탐색 트리에서 키 값이 최소인 노드 탐색

- 루트 노드로부터 왼쪽 자식 노드를 루트로 하는 서브 트리를 재귀적으로 반복 탐색 하여 왼쪽 자식 노드가 존재하지 않는 노드 Nmin을 반환 (i.e., **루트 노드로부터 왼쪽 자식 노드를 따라 내려가며, None을 만났을 때 None의 부모 노드를 반환**)

이진 탐색 트리에서 키 값이 최소인 노드 삭제

- Nmin을 탐색 후, Nmin의 부모 노드 Np와 Nmin의 자식 노드 Nc를 연결 시킴(Note: Nmin의 왼쪽 자식 노드는 존재할 수 없음 **왜?**)

예: 키 값이 최소인 노드 탐색 및 삭제



N4의 왼쪽 자식 노드가 None이므로 키 값이 최소인 노드는 N4

N4의 부모 노드 N2와 N4의 오른쪽 자식 노드 N8을 연결 시킴

이진 탐색 트리 (12/22)

❑ 이진 탐색 트리의 구현 contd.

- **find_min() 시간복잡도: O(h), 여기서 h는 트리의 높이(일반적으로 O(logN), 편향된 이진 탐색 트리의 경우 O(N))**

```
27      def find_min(self):
28          if self.root == None:
29              return None
30          return self._find_min(self.root)
31      def _find_min(self, n):
32          if n.get_left() == None:
33              return n
34          return self._find_min(n.get_left())
```

  - 키 값이 최소인 노드 검색
    - **라인 28-29: if** 루트 노드가 None이면(i.e., empty 이진 탐색 트리라면), None 반환
    - **라인 30:** protected 멤버함수 _find_min() 호출 (인자는 **루트 노드**)
  - **_find_min(self, n)**
    - **라인 32-33**: **if** 루트 노드 n의 왼쪽 자식 노드가 None이라면, **n의 키 값이 최소므로** n을 반환
    - **라인 34:** 루트 노드 n의 왼쪽 자식 노드가 None이 아니라면, 왼쪽 자식을 루트 노드로 하는 서브 트리에서 탐색 수행

이진 탐색 트리 (13/22)

❑ 이진 탐색 트리의 구현 contd.

- **delete_min()시간복잡도: O(h), 여기서 h는 트리의 높이(일반적으로 O(logN), 편향된 이진 탐색 트리의 경우 O(N))**

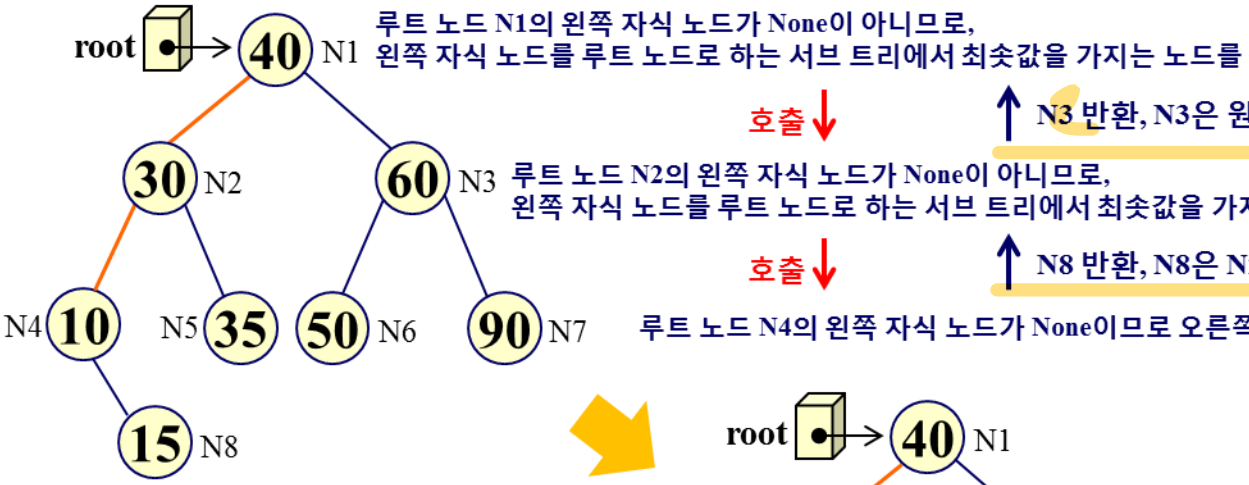
```
36      def delete_min(self):
37          if self.root == None:
38              print("Tree is empty.")
39          self.root = self._delete_min(self.root)
40      def _delete_min(self, n):
41          if n.get_left() == None:
42              return n.get_right()
43          n.set_left(self._delete_min(n.get_left()))
44          return n
```

  - 키 값이 최소인 노드 삭제
    - **라인 37-38: if** 루트 노드가 None이면(i.e., empty 이진 탐색 트리라면), "Tree is empty." 출력
    - **라인 30:** protected 멤버함수 _delete_min() 호출 (인자는 **루트 노드**)
  - **_delete_min(self, n)**
    - **라인 41-42: if** 루트 노드 n의 왼쪽 자식 노드가 None이라면, **n의 오른쪽 자식 노드**를 반환 (**Note:** 오른쪽 자식 노드가 없다면 **None을 반환**하게 됨)
    - **라인 43:** 루트 노드 n의 왼쪽 자식 노드가 None이 아니라면, 왼쪽 자식을 루트 노드로 하는 서브 트리에서 최솟값 삭제 연산 수행
    - **라인 44:** 부모 노드와 연결하기 위해 (현재 서브 트리의) 루트 노드 n 반환

이진 탐색 트리 (14/22)

❑ 이진 탐색 트리의 구현 contd.

예: 키 값이 최소인 노드 삭제



루트 노드 N1의 왼쪽 자식 노드가 None이 아니므로,
왼쪽 자식 노드를 루트 노드로 하는 서브 트리에서 최솟값을 가지는 노드를 삭제하기 위해 _delete_min(N2) 호출

호출 ↓

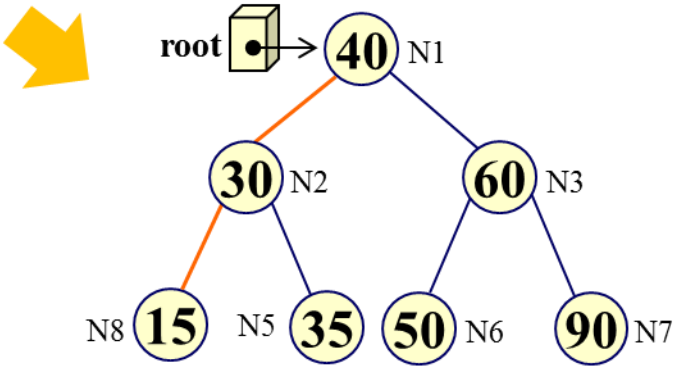
↑ N3 반환, N3은 원래대로 N1의 왼쪽 자식 노드가 됨

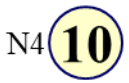
루트 노드 N2의 왼쪽 자식 노드가 None이 아니므로,
왼쪽 자식 노드를 루트 노드로 하는 서브 트리에서 최솟값을 가지는 노드를 삭제하기 위해 _delete_min(N4) 호출

호출 ↓

↑ N8 반환, N8은 N2의 왼쪽 자식 노드가 됨

루트 노드 N4의 왼쪽 자식 노드가 None이므로 오른쪽 자식 노드 N8을 반환





N4는 트리에서 분리 됨

```
36    def delete_min(self):
37        if self.root == None:
38            print("Tree is empty.")
39        self.root = self._delete_min(self.root)
40    def _delete_min(self, n):
41        if n.get_left() == None:
42            return n.get_right()
43        n.set_left(self._delete_min(n.get_left()))
44        return n
```

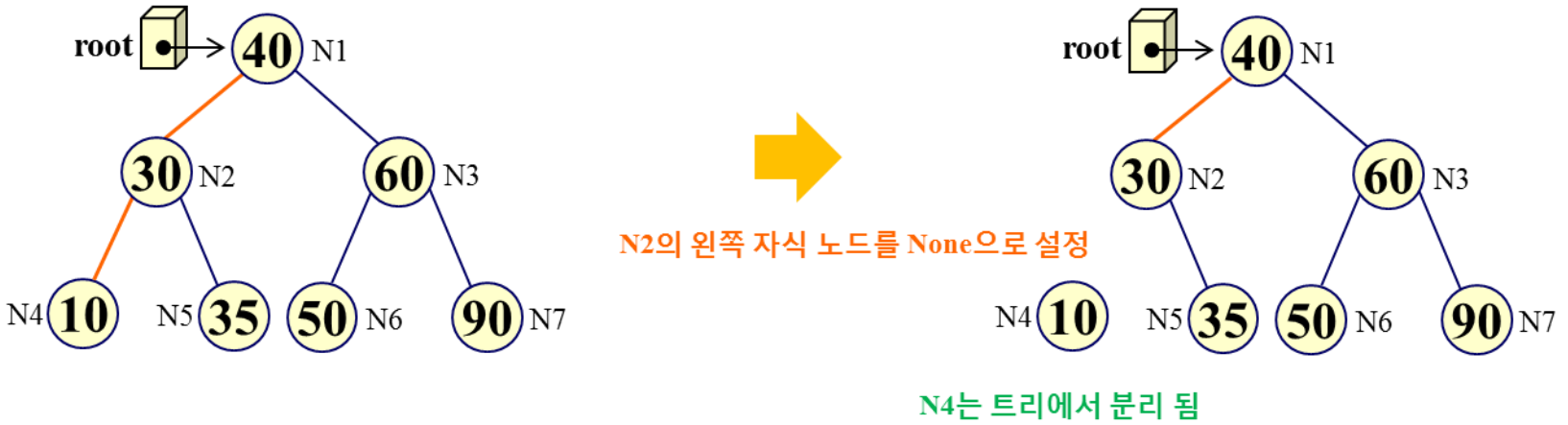
이진 탐색 트리 (15/22)

❑ 이진 탐색 트리의 구현 contd.

-이진 탐색 트리에서 특정 키 값 key를 가지는 노드 삭제-

- 삭제하고자 하는 노드 N을 찾고 삭제 후, 이진 탐색 트리의 조건을 만족하도록 N의 부모 노드 Np와 N의 자식 노드(들) Nc를 연결
- N이 자식 노드가 없는 경우(Case 0), 자식 노드가 하나인 경우(Case 1), 자식 노드가 둘인 경우(Case 2)로 나누어 삭제 연산을 수행
  - **Case 0**: Np의 (left 혹은 right) 링크 필드에서 N의 참조를 None으로 설정
  - **Case 1**: Np와 Nc를 연결
  - **Case 2:** Np는 하나인데 Nc는 둘이므로 Case 1의 방법을 사용할 수 없음. 따라서 N의 위치에 트리를 중위 순회하면서 N을 방문하기 직전 노드(중위 선행자: N의 왼쪽 서브 트리에서 가장 큰 키 값을 가지는 자손 노드) 또는 직후에 방문되는 노드(중위 후속자: N의 오른쪽 서브 트리에서 가장 작은 키 값을 가지는 자손 노드)로 대체

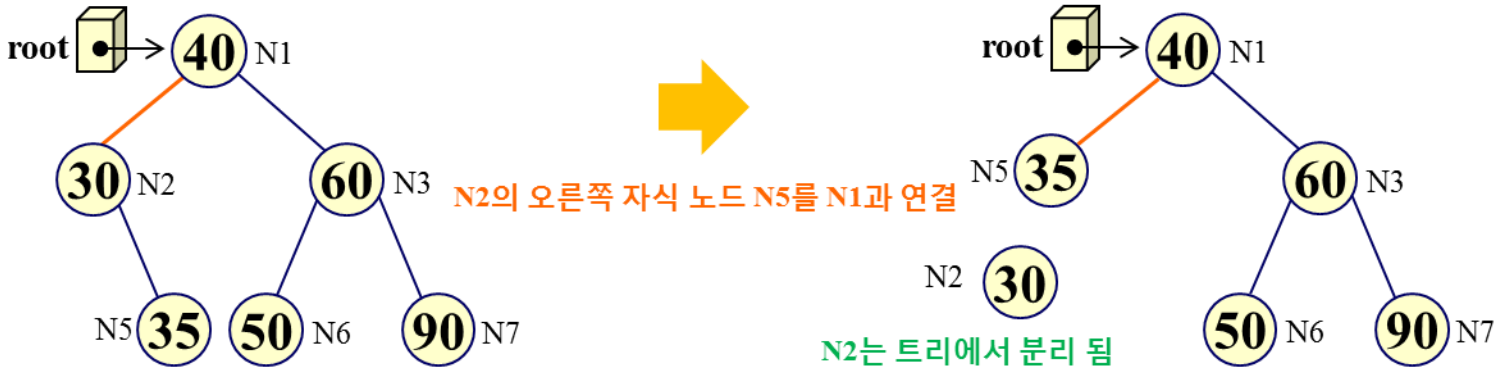
Case 0의 예: 키 값이 10인 노드 N4 삭제



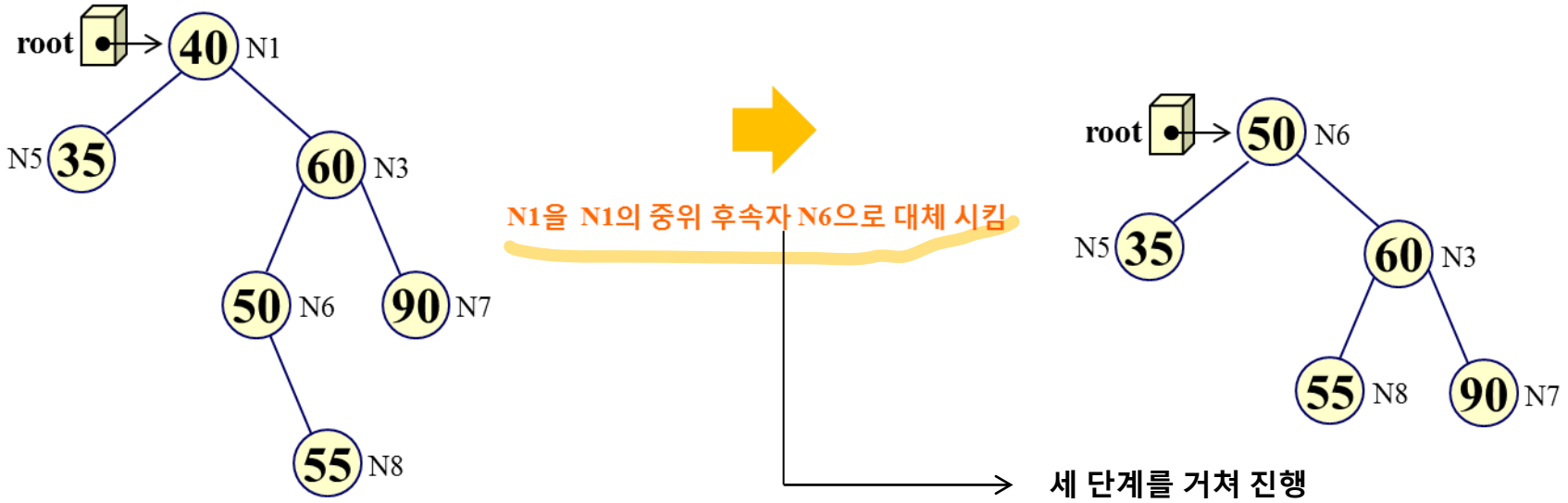
이진 탐색 트리 (16/22)

❑ 이진 탐색 트리의 구현 contd.

Case 1의 예: 키 값이 30인 노드 N2 삭제



Case 2의 예: 키 값이 40인 노드 N1 삭제



이진 탐색 트리 (17/22)

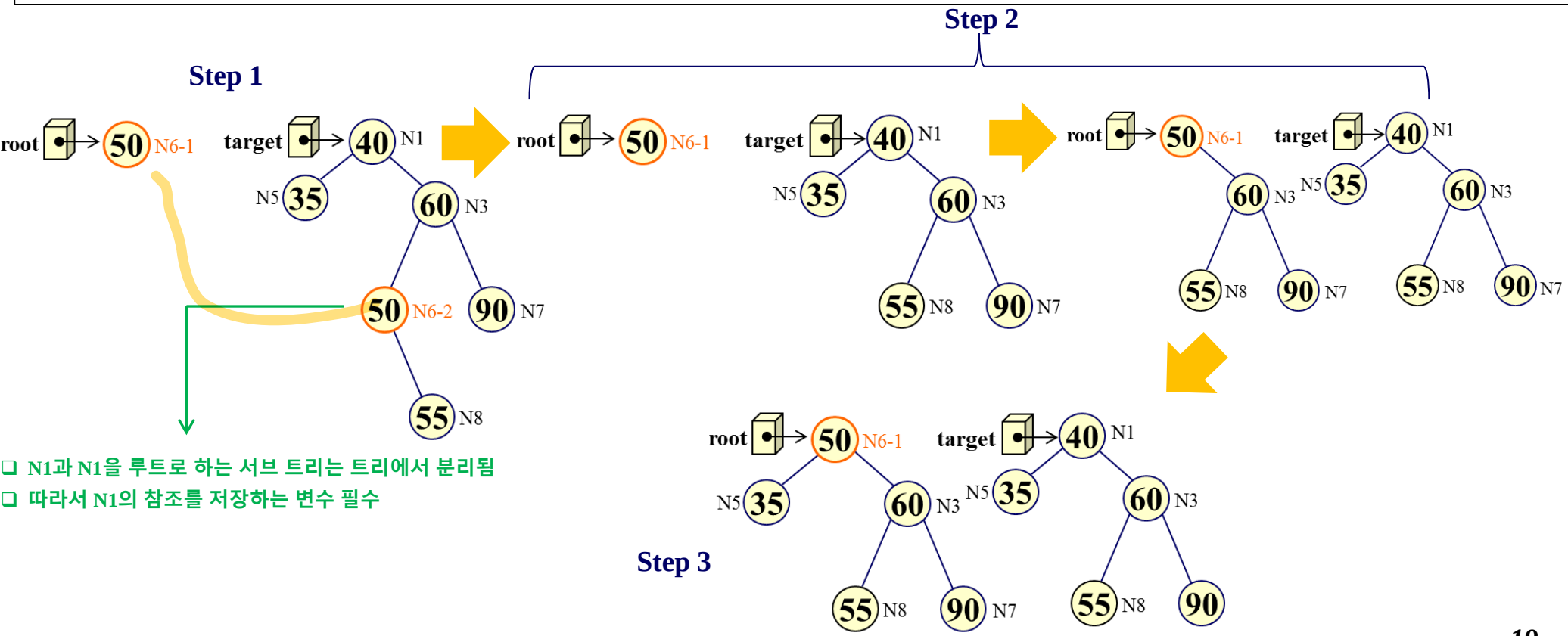
❑ 이진 탐색 트리의 구현 contd.

Case 2의 예: 키 값이 40인 노드 N1 삭제 contd.

Step 1: N1의 오른쪽 자식 노드 N3에 대해 _find_min(N3)을 호출하여, 루트 노드를 N3으로 하는 서브 트리에서 키 값이 최소인 노드 N6을 찾아서 N1을 N6으로 대체
(현재 N6은 두 개로 N6-1, N6-2로 표기하며 N1과 N1을 루트로 하는 서브 트리는 트리에서 분리)

Step 2: (1) N6-2를 삭제하기 위해 N1의 오른쪽 자식 노드 N3에 대해 _delete_min(N3)을 호출하여 N6-2를 삭제 후, (2) N3을 N6-1의 오른쪽 자식으로 연결함
(N3을 N6-1의 오른쪽 자식으로 연결하였으므로 N3을 루트로 하는 서브 트리의 모든 노드들이 N6-1의 후손 노드가 됨)

Step 3: N1의 왼쪽 자식 노드 N5를 N6-1의 왼쪽 자식 노드로 연결함
(N5를 N6-1의 왼쪽 자식으로 연결하였으므로 N5을 루트로 하는 서브 트리의 모든 노드들이 N6-1의 후손 노드가 됨)



- ❑ N1과 N1을 루트로 하는 서브 트리는 트리에서 분리됨
- ❑ 따라서 N1의 참조를 저장하는 변수 필수

이진 탐색 트리 (18/22)

❑ 이진 탐색 트리의 구현 contd.

- **delete(key)** **시간복잡도: O(h), 여기서 h는 트리의 높이(일반적으로 O(logN), 편향된 이진 탐색 트리의 경우 O(N))**

```
46      def delete(self, key):
47          self.root = self._delete(self.root, key)
48      def _delete(self, n, key):
49          if n == None:
50              return None
51          if n.get_key() > key:
52              n.set_left(self._delete(n.get_left(), key))
53          elif n.get_key() < key:
54              n.set_right(self._delete(n.get_right(), key))
55          else:
56              if n.get_left() == None and n.get_right() == None:
57                  return None
58              if n.get_left() == None or n.get_right() == None:
59                  if n.get_left() == None:
60                      return n.get_right()
61                  else:
62                      return n.get_left()
63              target = n
64              n = self._find_min(target.get_right())
65              n.set_right(self._delete_min(target.get_right()))
66              n.set_left(target.get_left())
67          return n
```

  - 키 값 key를 가지는 노드 삭제
    - **라인 47:** protected 멤버함수 _delete() 호출 (인자는 **루트 노드와 키 값 key**)
  - **_delete(self, n, key)**
    - **라인 49-50**: **if** 루트 노드 n이 None이면 None 반환(삭제할 노드가 없는 경우)

이진 탐색 트리 (19/22)

❑ 이진 탐색 트리의 구현 contd.

- **delete(key) contd.**

```
46      def delete(self, key):
47          self.root = self._delete(self.root, key)
48      def _delete(self, n, key):
49          if n == None:
50              return None
51          if n.get_key() > key:
52              n.set_left(self._delete(n.get_left(), key))
53          elif n.get_key() < key:
54              n.set_right(self._delete(n.get_right(), key))
55          else:
56              if n.get_left() == None and n.get_right() == None:
57                  return None
58              if n.get_left() == None or n.get_right() == None:
59                  if n.get_left() == None:
60                      return n.get_right()
61                  else:
62                      return n.get_left()
63              target = n
64              n = self._find_min(target.get_right())
65              n.set_right(self._delete_min(target.get_right()))
66              n.set_left(target.get_left())
67          return n
```

  - **_delete(self, n, key)**
    - **라인 51-52: if** 루트 노드 n의 키 값이 key보다 크면 n의 왼쪽 자식 노드를 루트로 하는 서브 트리에서 삭제할 노드 탐색
    - **라인 53-54: elif** 루트 노드 n의 키 값이 key보다 작으면 n의 오른쪽 자식 노드를 루트로 하는 서브 트리에서 삭제할 노드 탐색

이진 탐색 트리 (20/22)

❑ 이진 탐색 트리의 구현 contd.

- **delete(key) contd.**

```
46      def delete(self, key):
47          self.root = self._delete(self.root, key)
48      def _delete(self, n, key):
49          if n == None:
50              return None
51          if n.get_key() > key:
52              n.set_left(self._delete(n.get_left(), key))
53          elif n.get_key() < key:
54              n.set_right(self._delete(n.get_right(), key))
55          else:
56              if n.get_left() == None and n.get_right() == None:
57                  return None
58              if n.get_left() == None or n.get_right() == None:
59                  if n.get_left() == None:
60                      return n.get_right()
61                  else:
62                      return n.get_left()
63              target = n
64              n = self._find_min(target.get_right())
65              n.set_right(self._delete_min(target.get_right()))
66              n.set_left(target.get_left())
67          return n
```

  - **_delete(self, n, key)**
    - **라인 55: else**(키 값 key를 가지는 노드를 찾았다면),
      - » **라인 56-57**: Case 0인 경우, None을 반환
      - » **라인 58-62:** Case 1인 경우(왼쪽 자식 노드 혹은 오른쪽 자식 노드가 None인 경우) 중 왼쪽 자식 노드가 None이면 오른쪽 자식 노드를 n의 부모와 연결하고, 오른쪽 자식 노드가 None이면 왼쪽 자식 노드를 n의 부모와 연결

이진 탐색 트리 (21/22)

❑ 이진 탐색 트리의 구현 contd.

- **delete(key) contd.**

```
46      def delete(self, key):
47          self.root = self._delete(self.root, key)
48      def _delete(self, n, key):
49          if n == None:
50              return None
51          if n.get_key() > key:
52              n.set_left(self._delete(n.get_left(), key))
53          elif n.get_key() < key:
54              n.set_right(self._delete(n.get_right(), key))
55          else:
56              if n.get_left() == None and n.get_right() == None:
57                  return None
58              if n.get_left() == None or n.get_right() == None:
59                  if n.get_left() == None:
60                      return n.get_right()
61                  else:
62                      return n.get_left()
63              target = n
64              n = self._find_min(target.get_right())
65              n.set_right(self._delete_min(target.get_right()))
66              n.set_left(target.get_left())
67          return n
```

  - **_delete(self, n, key)**
    - **라인 55: else**(키 값 key를 가지는 노드를 찾았다면),
      - **라인 63-66:**: Case 2인 경우, 지역 변수 target을 생성하여 n을 참조하게 하고(**라인 63**), n의 중위 후속자를 찾아서 n으로 대체 시키고(**라인 64**), target 변수가 참조하고 있는 nold의 오른쪽 자식 cright를 루트로 하는 서브 트리에서 중위 후속자를 삭제 후 cright를 nnew의 오른쪽 자식으로 연결 시키고(**라인 65**), nold의 왼쪽 자식 cleft를 nnew의 왼쪽 자식으로 연결시킴(**라인 66**)

n의 중위 후속자가 n이 됨(이제부터 새로운 n을 nnew
이전 n을 nold라고 표기

이진 탐색 트리 (22/22)

❑ 이진 탐색 트리의 구현 contd.

- **delete(key) contd.**

```
46      def delete(self, key):
47          self.root = self._delete(self.root, key)
48      def _delete(self, n, key):
49          if n == None:
50              return None
51          if n.get_key() > key:
52              n.set_left(self._delete(n.get_left(), key))
53          elif n.get_key() < key:
54              n.set_right(self._delete(n.get_right(), key))
55          else:
56              if n.get_left() == None and n.get_right() == None:
57                  return None
58              if n.get_left() == None or n.get_right() == None:
59                  if n.get_left() == None:
60                      return n.get_right()
61                  else:
62                      return n.get_left()
63              target = n
64              n = self._find_min(target.get_right())
65              n.set_right(self._delete_min(target.get_right()))
66              n.set_left(target.get_left())
67          return n
```

  - **_delete(self, n, key)**
    - **라인 67:** 부모 노드와 연결하기 위해 (현재 서브 트리의) 루트 노드 n 반환(만약 Case 0인경우라면 라인 57에서 None을 반환하고, Case 1인 경우는 라인 60 혹은 라인 62에서 본인의 오른쪽 자식 노드 혹은 왼쪽 자식 노드를 반환하고, Case 2인 경우 경우 라인 67을 통해 반환