

COSE213: Data Structure

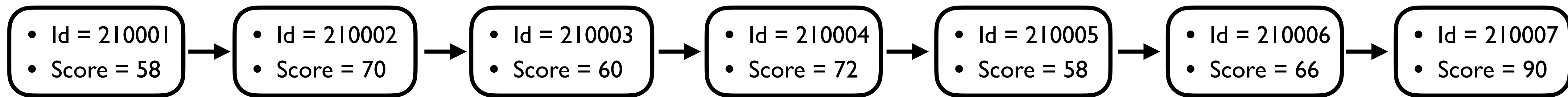
Lecture 8 - 이진 탐색 트리 (Binary Search Tree)

Minseok Jeon

2024 Fall

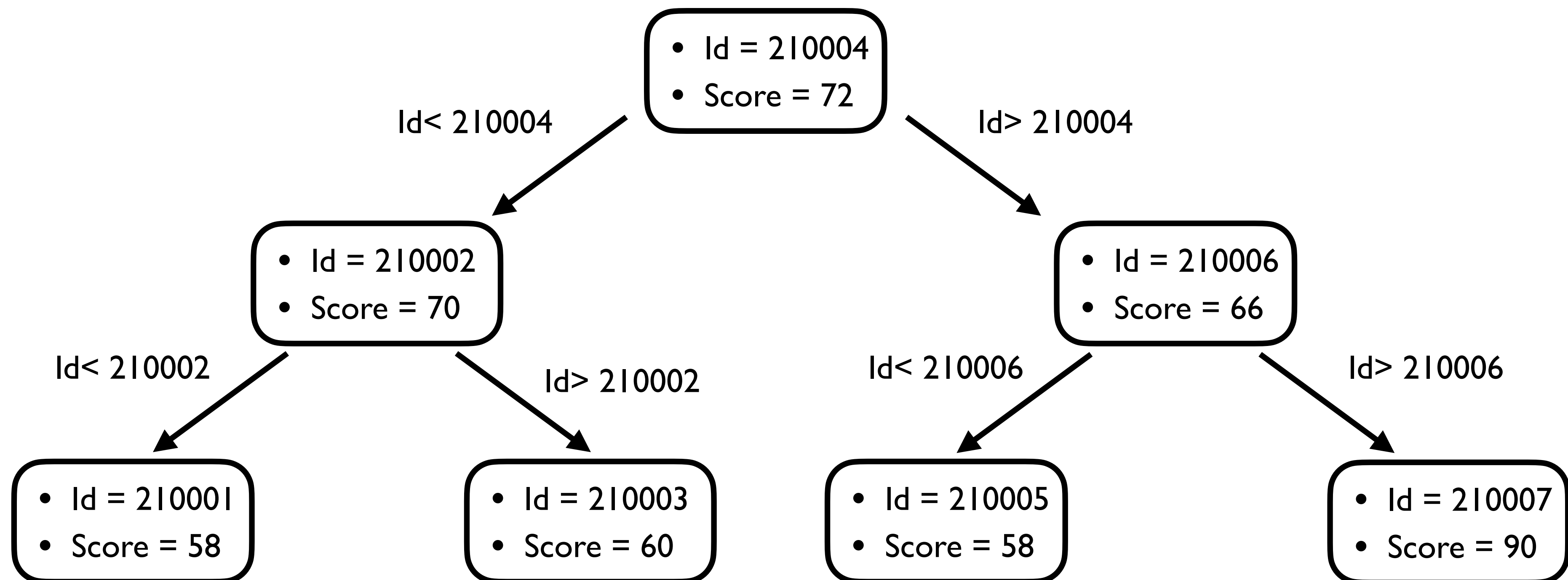
문제: 학생의 점수 조회

- 아래와 같이 학번(id)을 기준으로 정렬된 리스트에서 특정한 학번의 점수를 조회하고 싶은 경우
 - 최악의 경우 노드의 개수만큼 (7번) 조회를 해야함 (시간 복잡도 = $O(n)$)



문제: 학생의 점수 조회

- 아래와 같이 학번(id)을 기준으로 정렬된 리스트에서 특정한 학번의 점수를 조회하고 싶은 경우
- 문제 해결을 위한 아이디어: 데이터를 아래와 같이 저장하면 어떨까?
 - 최악의 경우에도 2번만에 원하는 노드를 찾을 수 있음

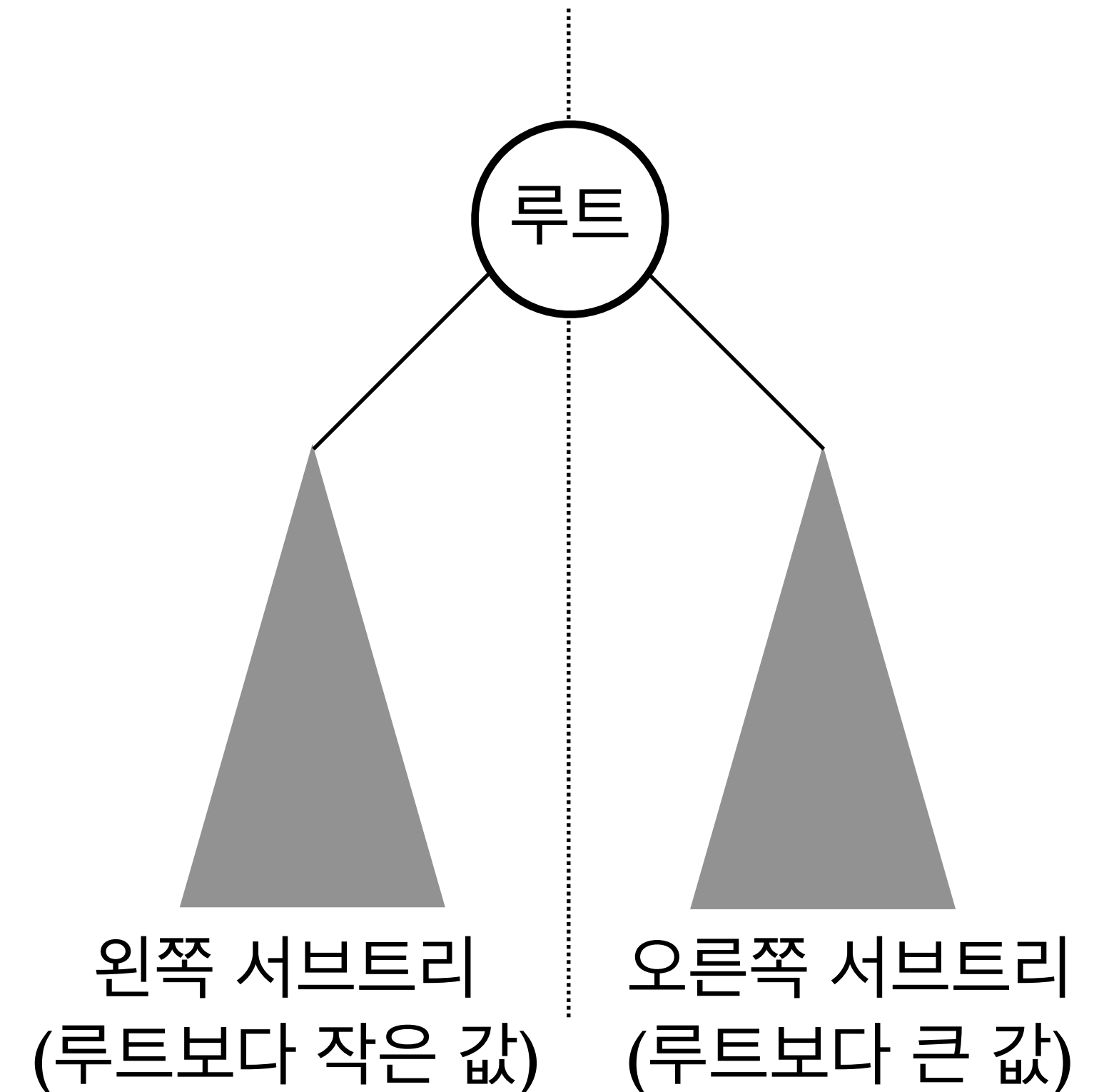


이진 탐색 트리 (Binary Search Tree)

- 이진 탐색 트리(BST, Binary Search Tree)는 이진트리 기반의 탐색을 위한 자료구조임
- 이진 탐색 트리는 다음과 같이 (재귀적으로) 정의됨

Definition

- (1) 모든 노드는 유일한 키(key)를 갖는다.
- (2) 왼쪽 서브트리의 키들은 루트의 키보다 작다.
- (3) 오른쪽 서브트리의 키들은 루트의 키보다 크다.
- (4) 왼쪽과 오른쪽 서브트리도 이진 탐색 트리이다.

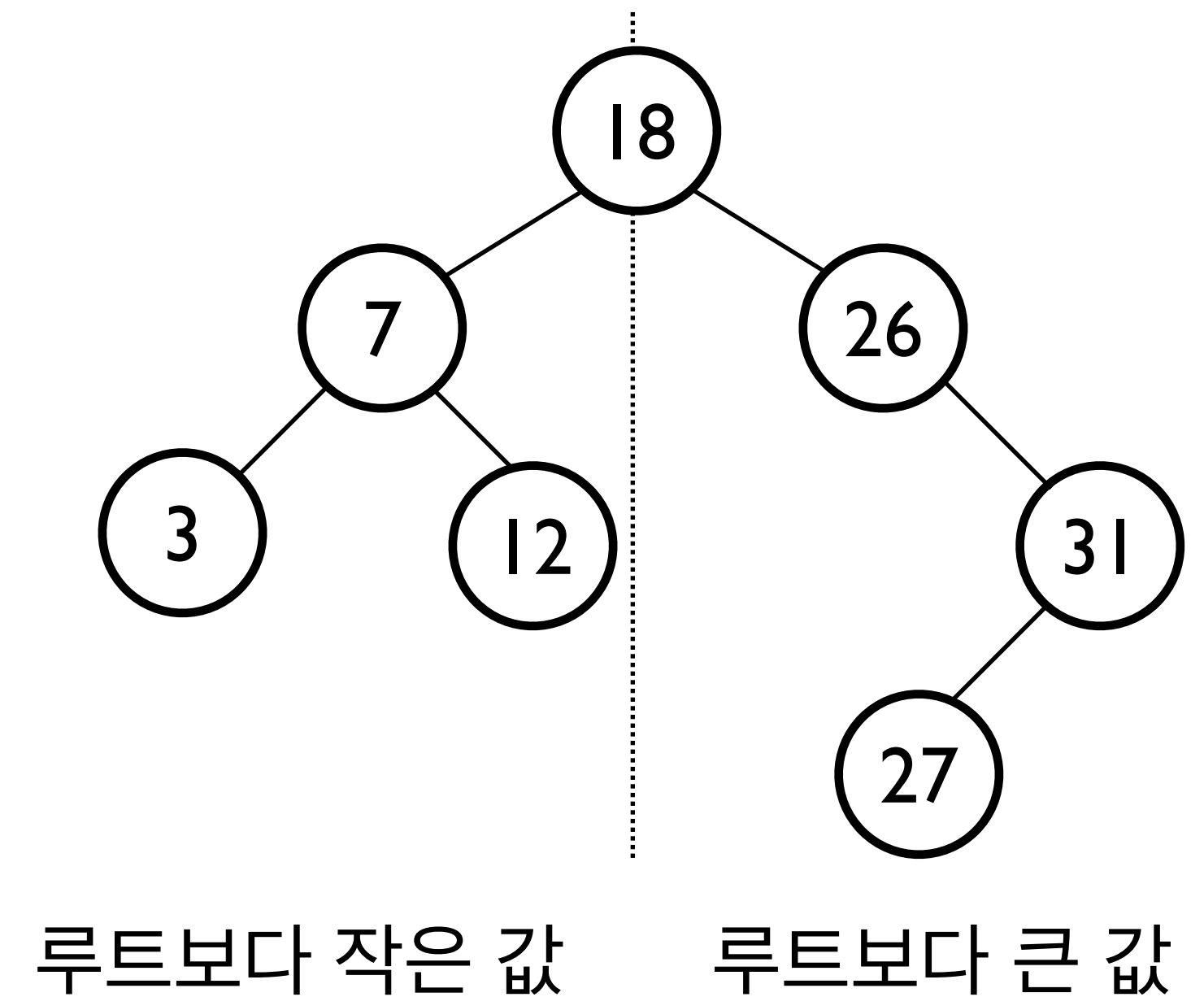


이진 탐색 트리 (Binary Search Tree)

- 이진 탐색 트리(BST, Binary Search Tree)는 이진트리 기반의 탐색을 위한 자료구조임
- 이진 탐색 트리는 다음과 같이 (재귀적으로) 정의됨

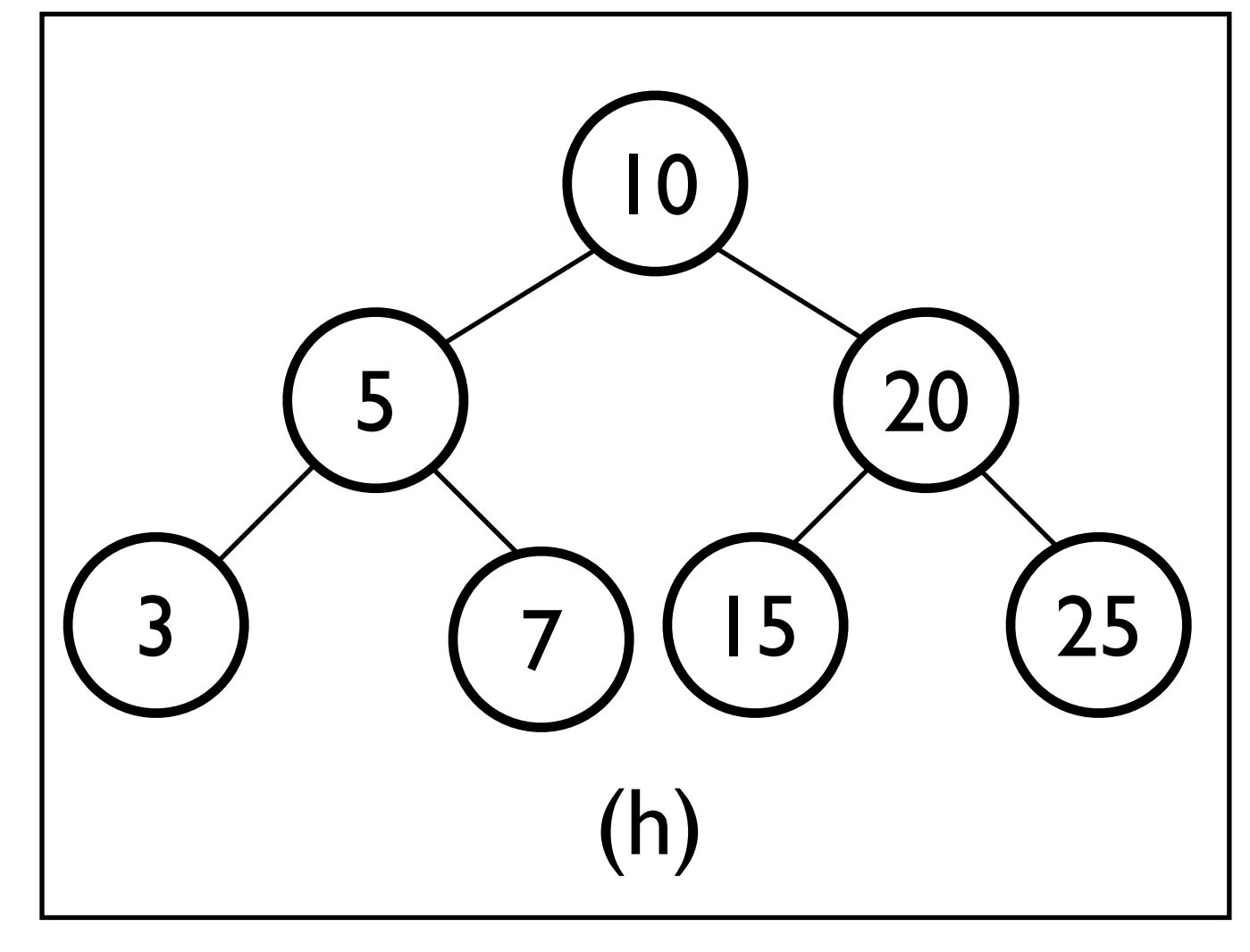
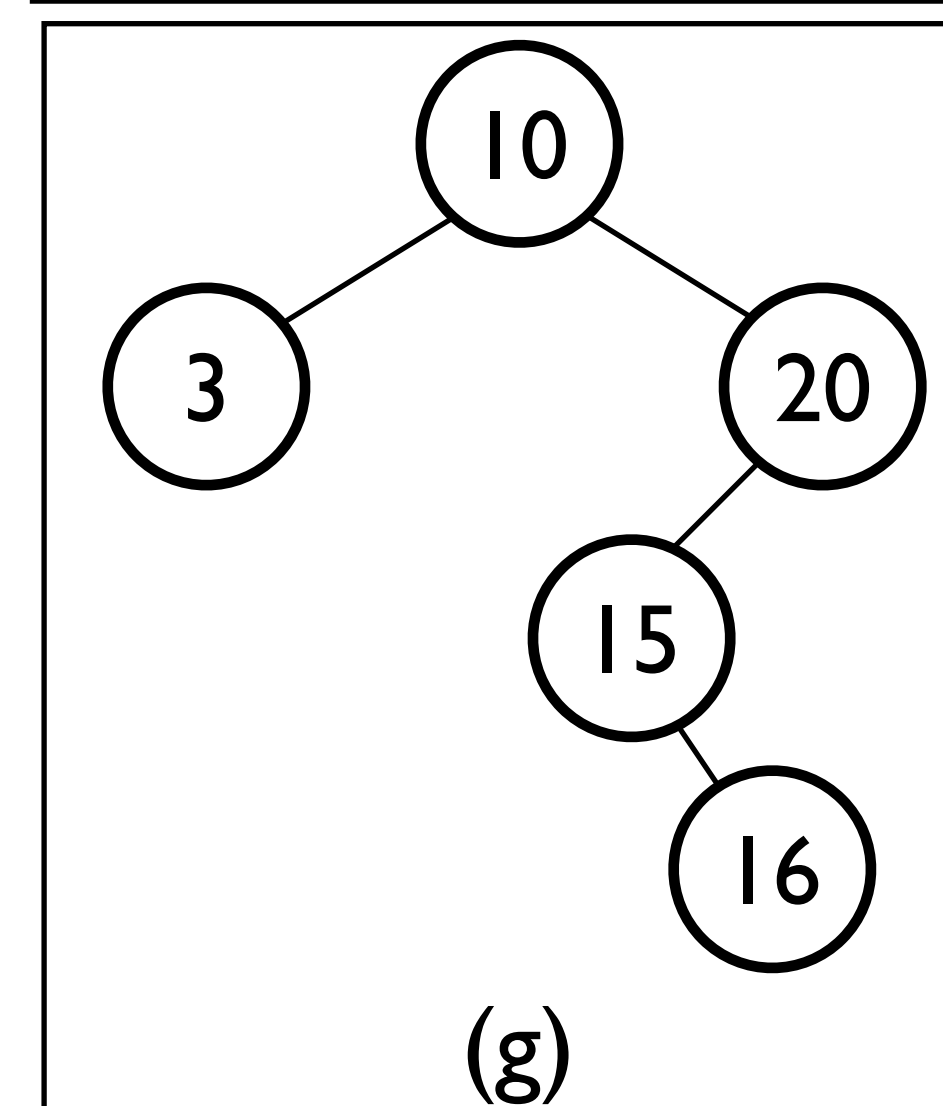
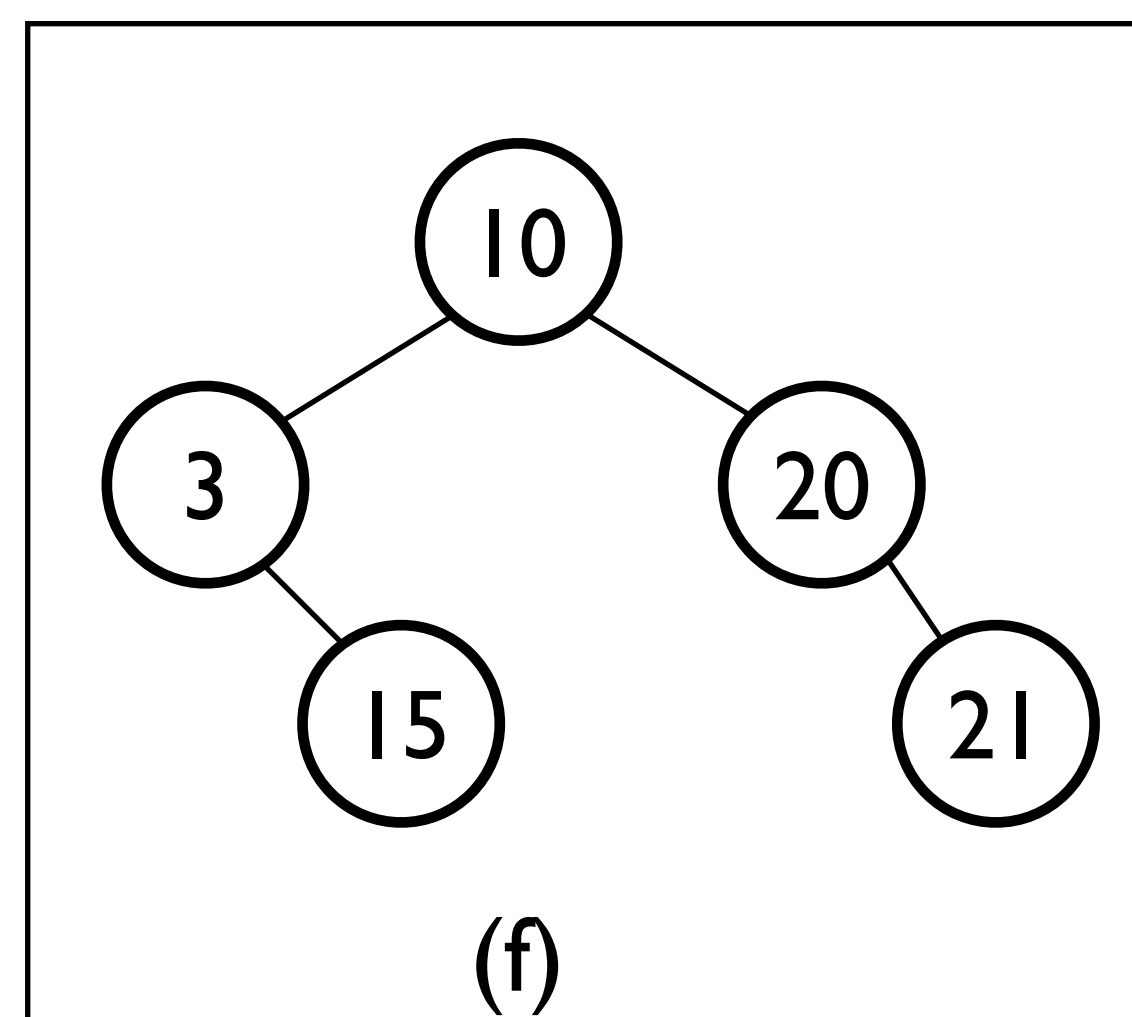
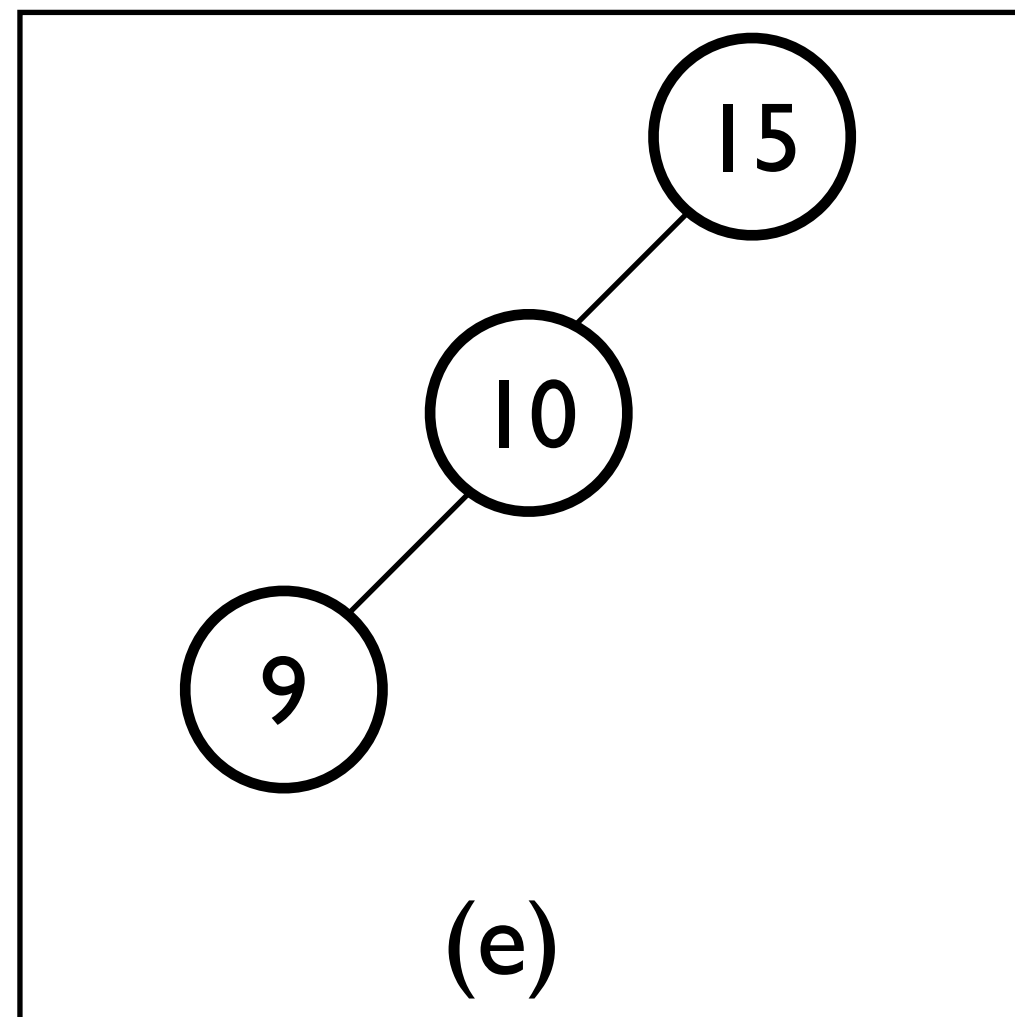
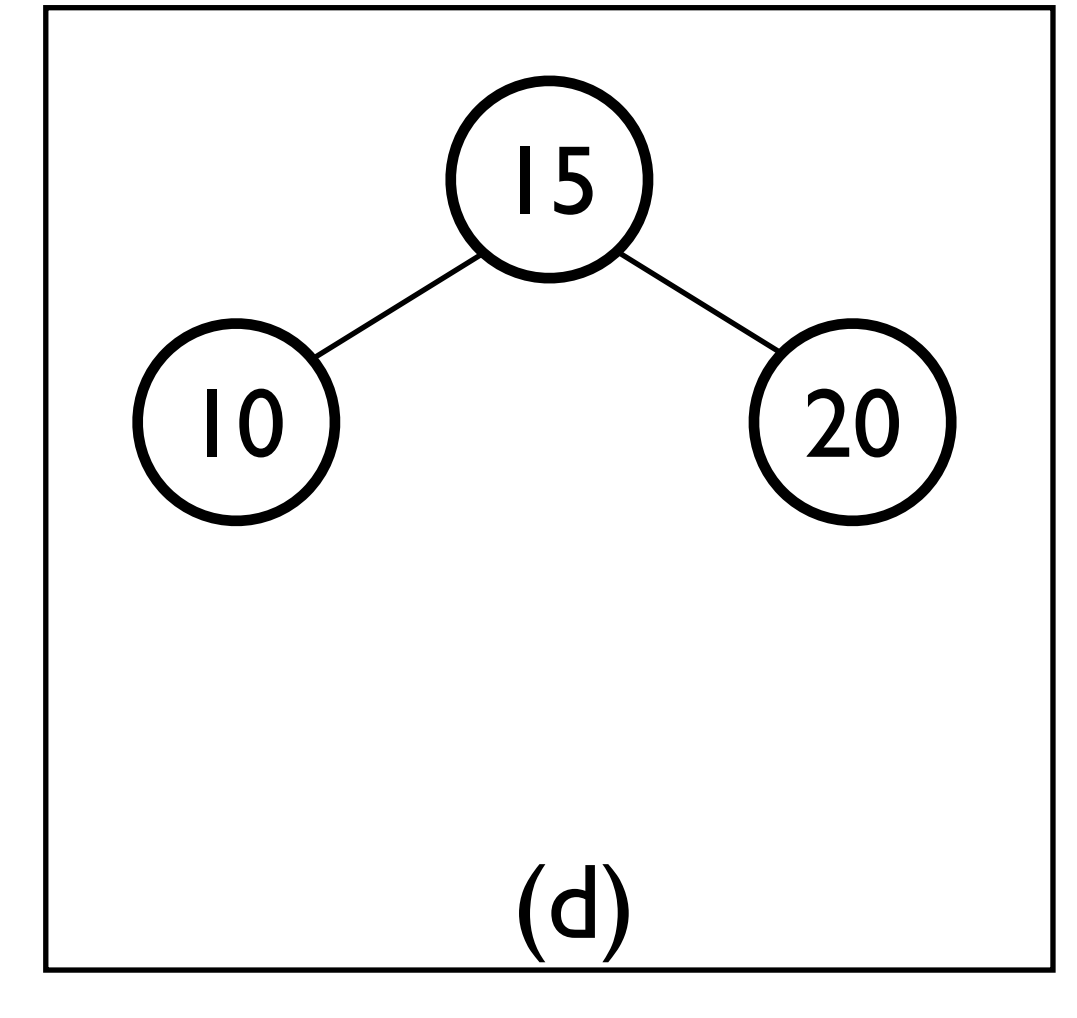
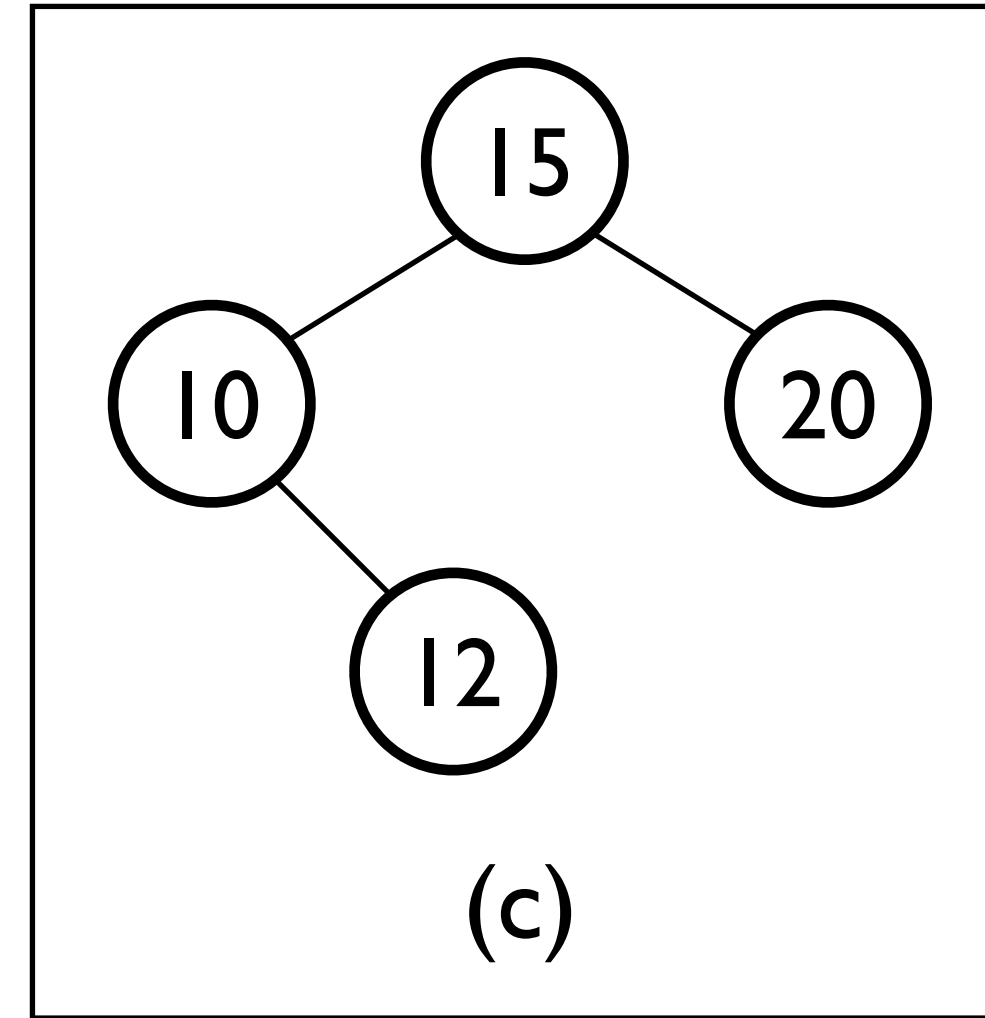
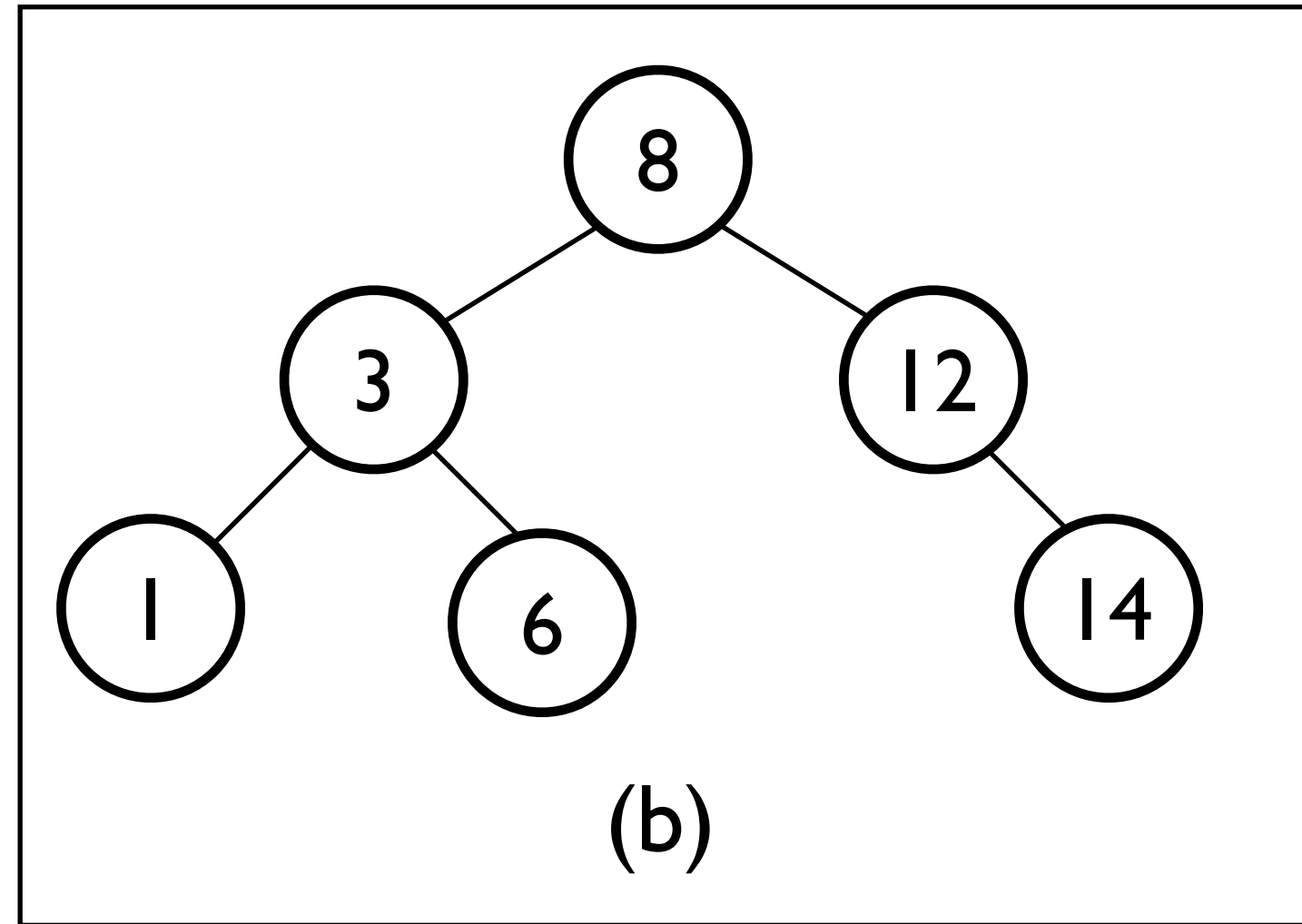
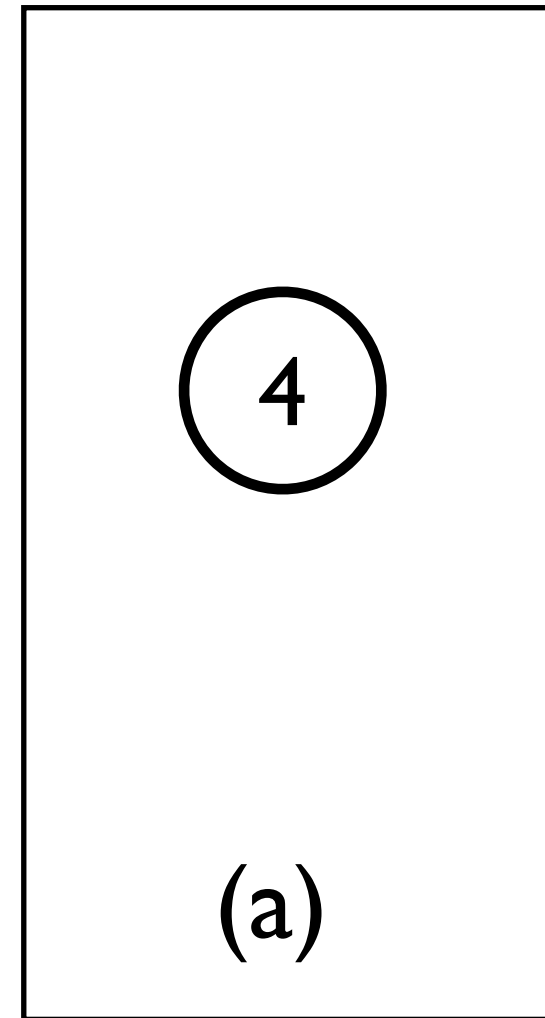
Definition

- (1) 모든 노드는 유일한 키(key)를 갖는다.
- (2) 왼쪽 서브트리의 키들은 루트의 키보다 작다.
- (3) 오른쪽 서브트리의 키들은 루트의 키보다 크다.
- (4) 왼쪽과 오른쪽 서브트리도 이진 탐색 트리이다.



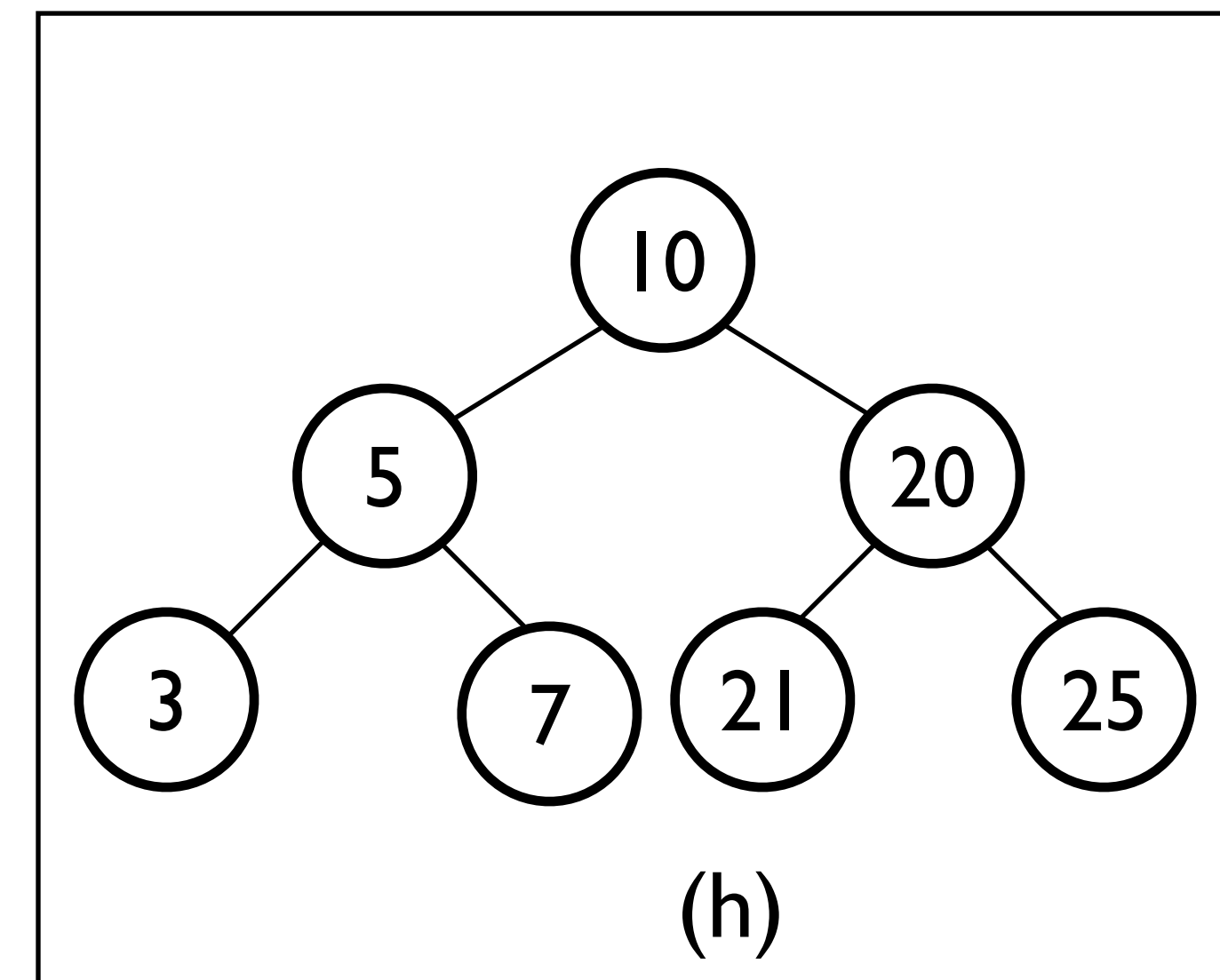
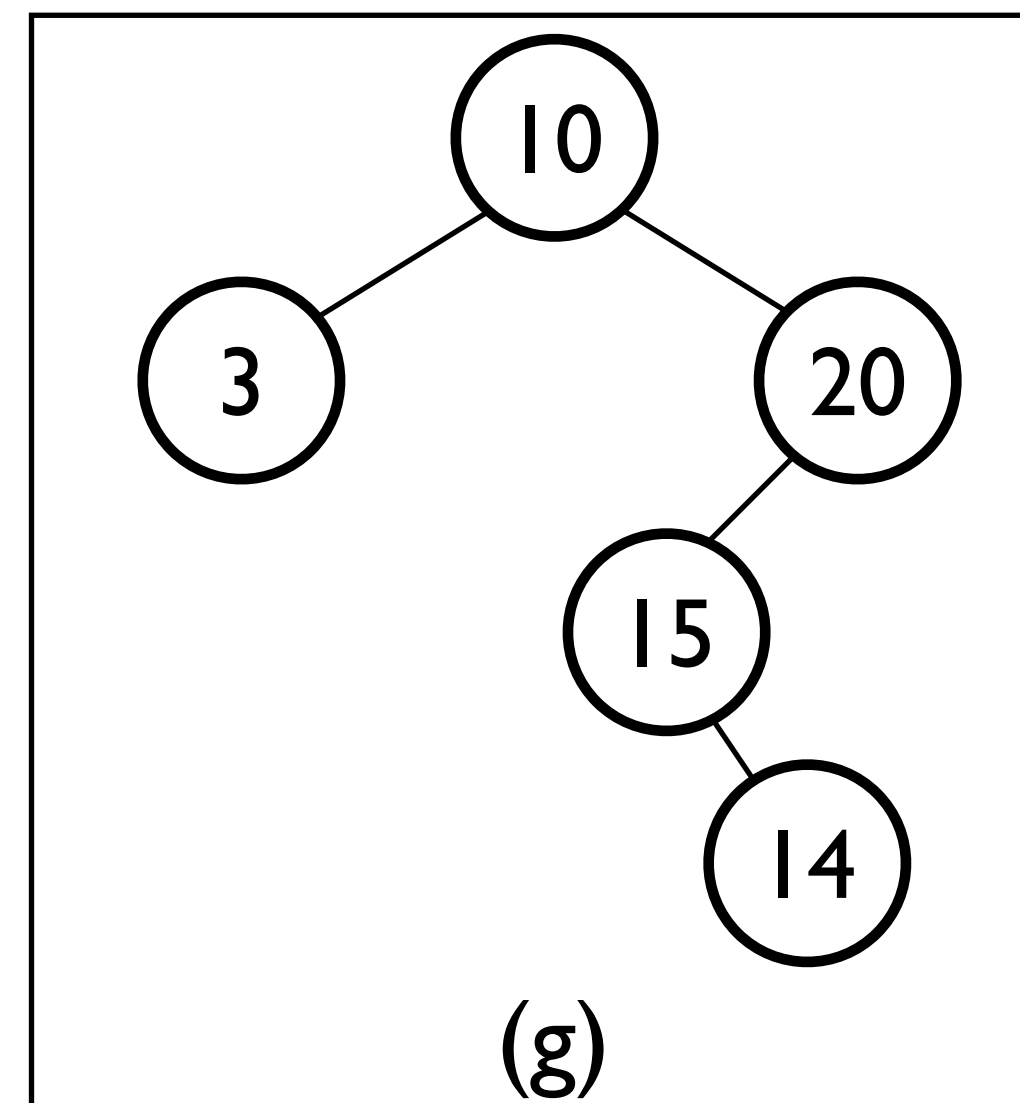
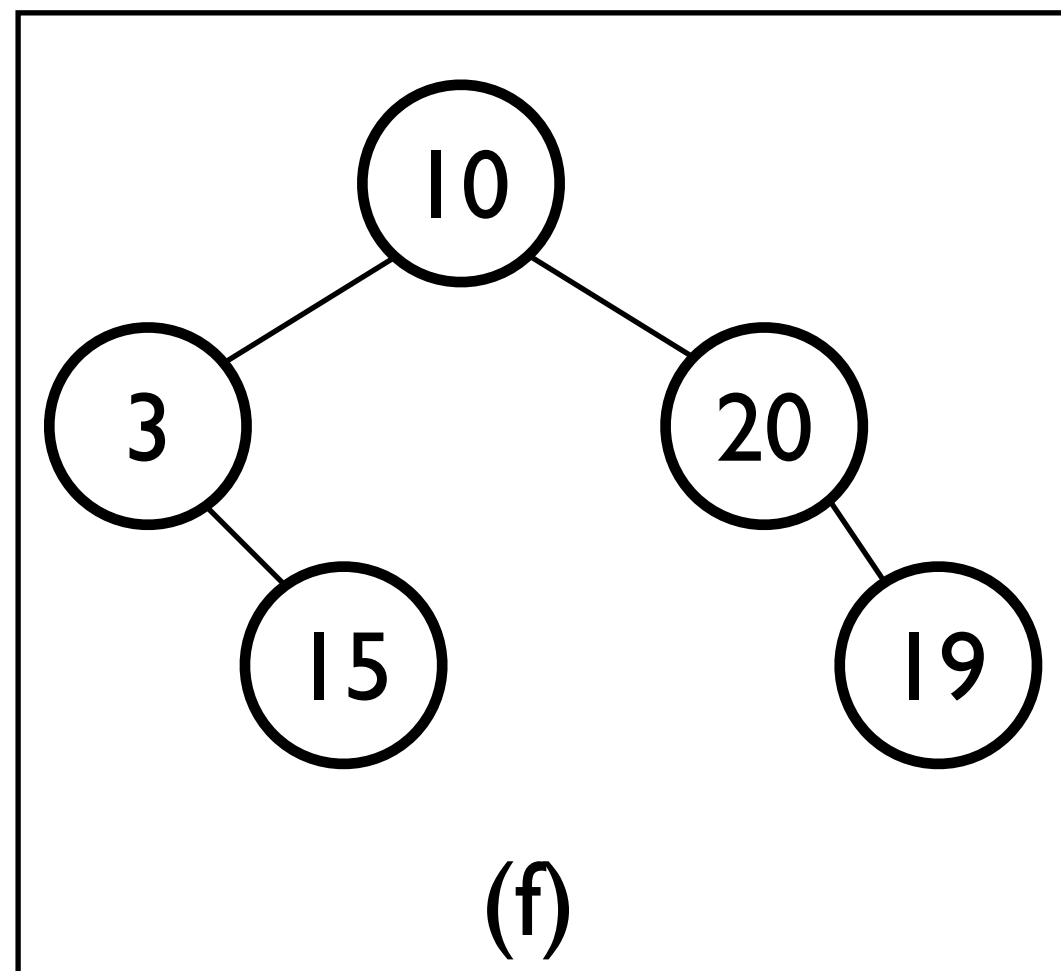
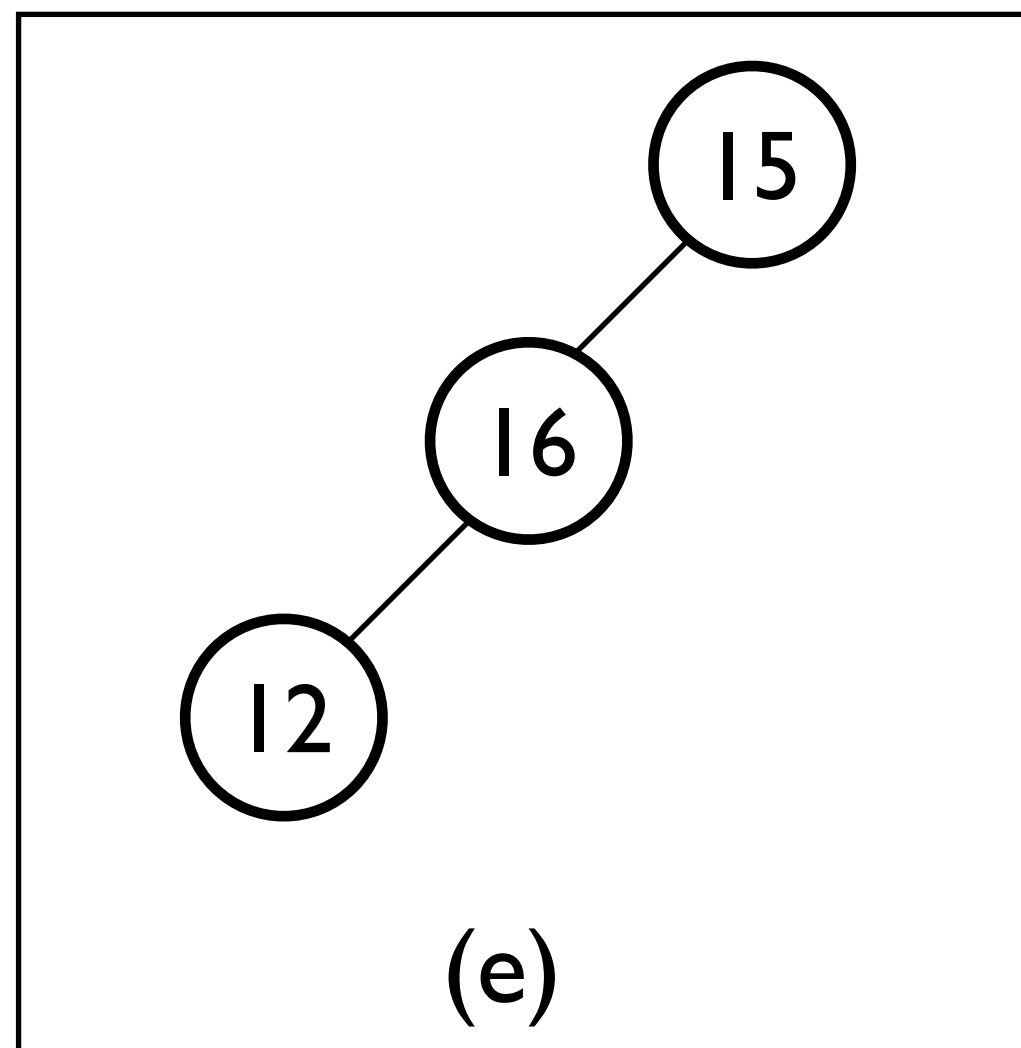
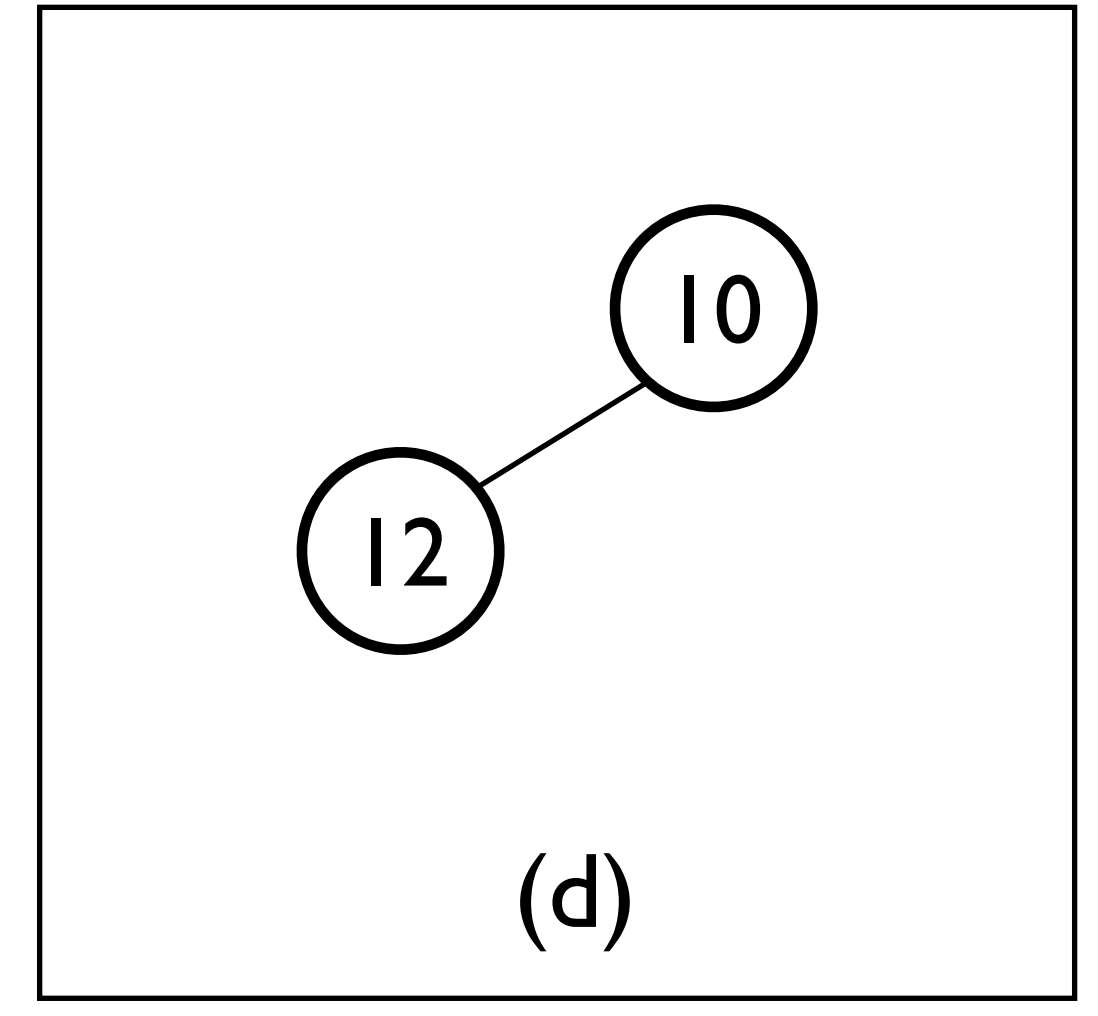
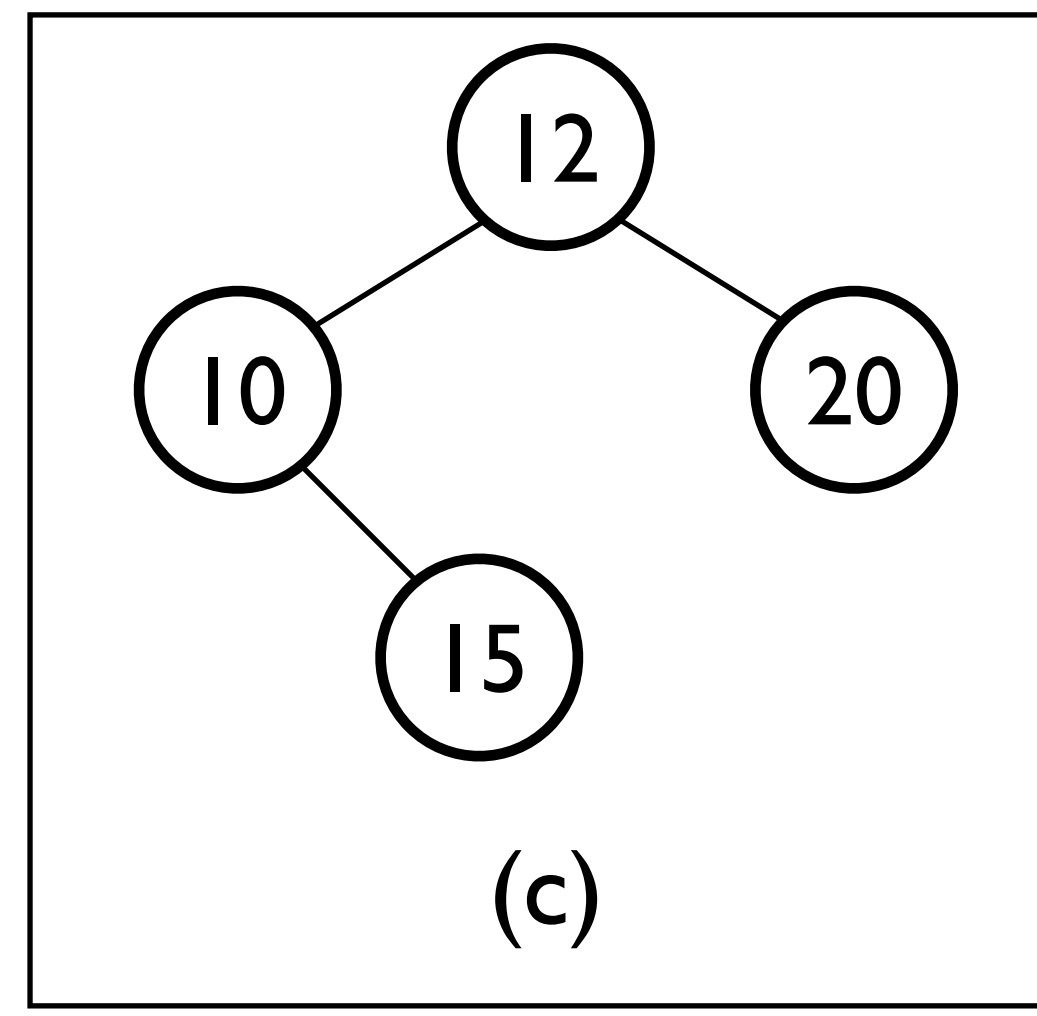
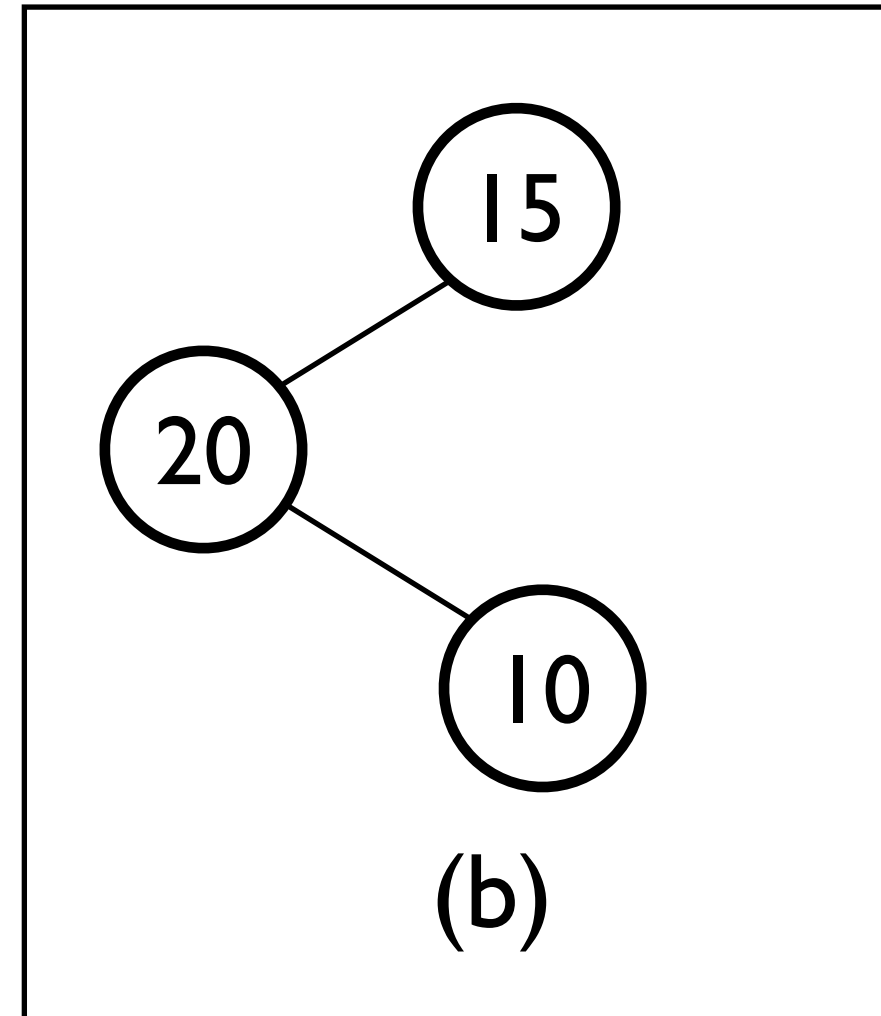
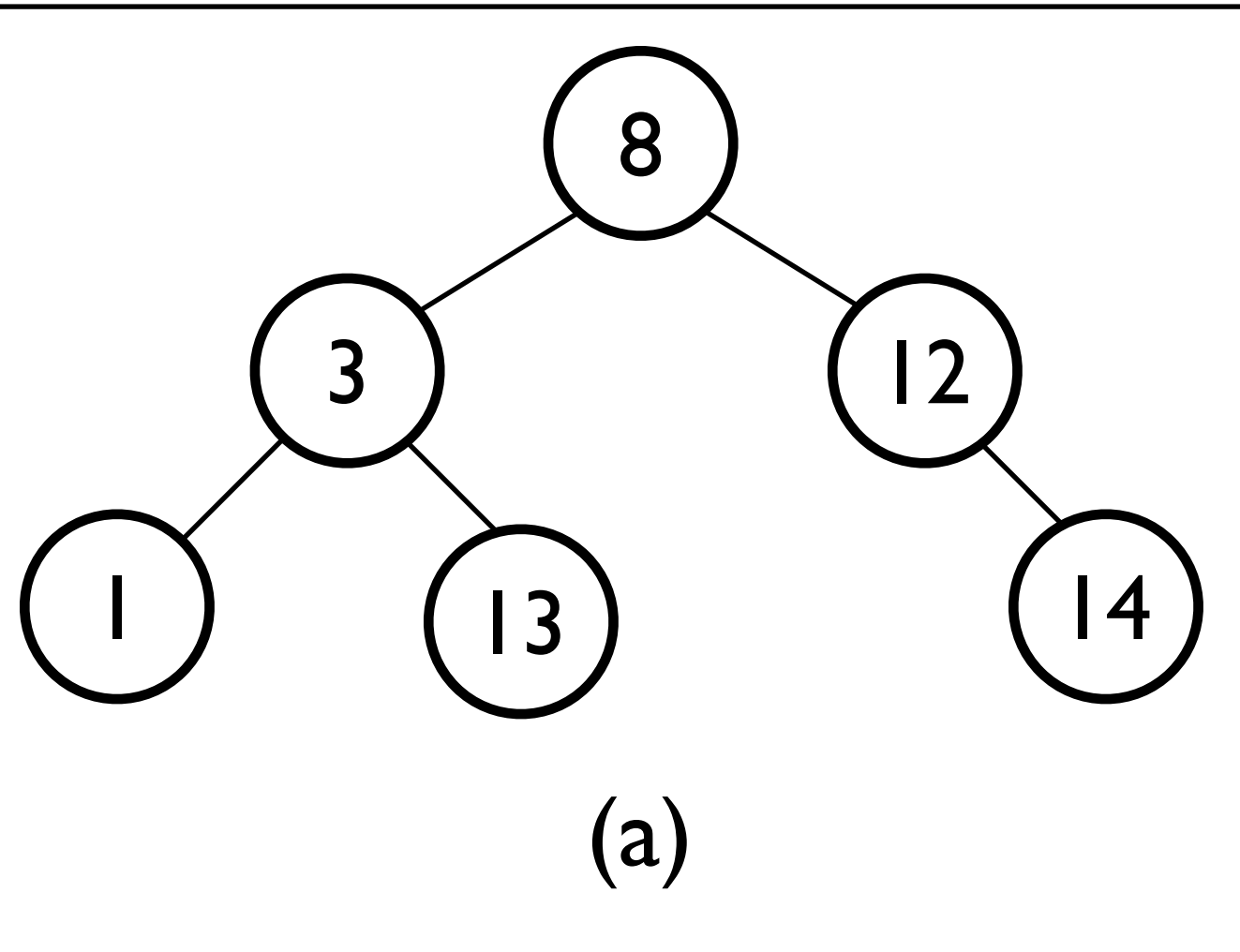
이진 탐색 트리 (Binary Search Tree)

- 유효한 이진 탐색 트리들



이진 탐색 트리 (Binary Search Tree)

- 유효하지 않은 이진 탐색 트리들



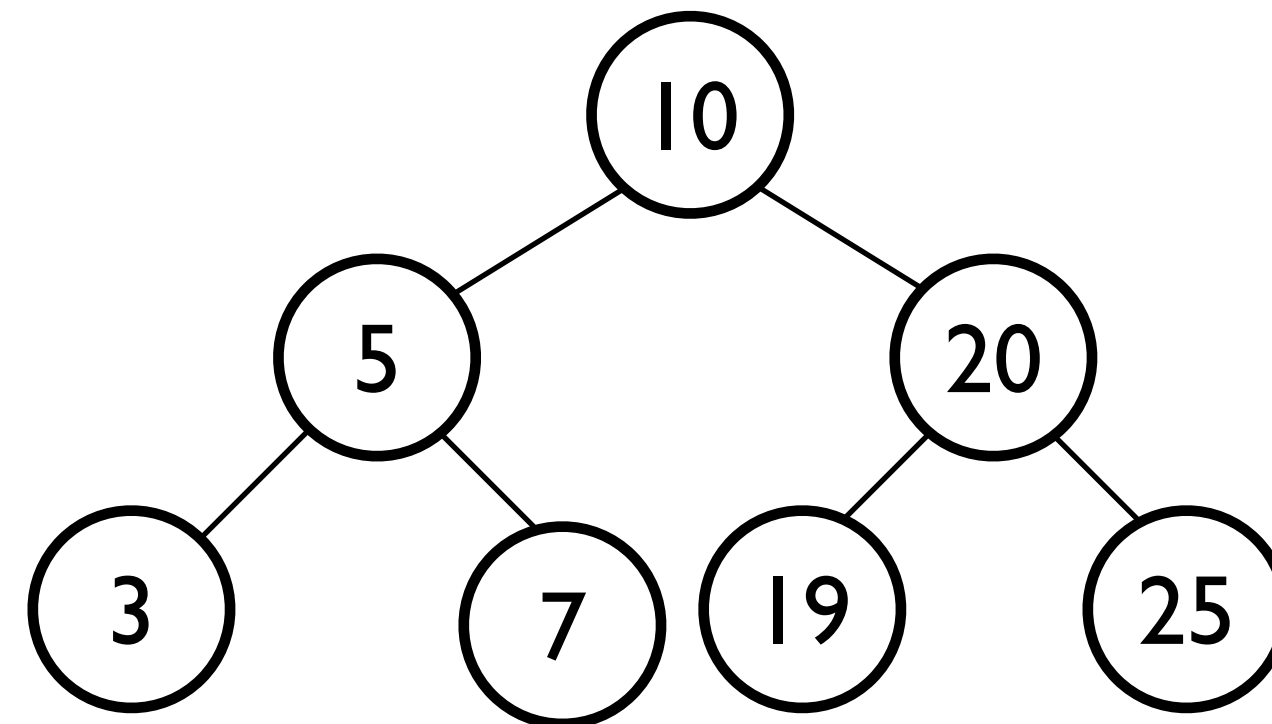
이진 탐색 트리의 순회 (Traversal)

- Example: 이진 탐색 트리의 키값들을 오름차순으로 출력하고싶다 어떻게 순회를 해야 할까?

전위순회(preorder)

중위순회(inorder)

후위순회(postorder)



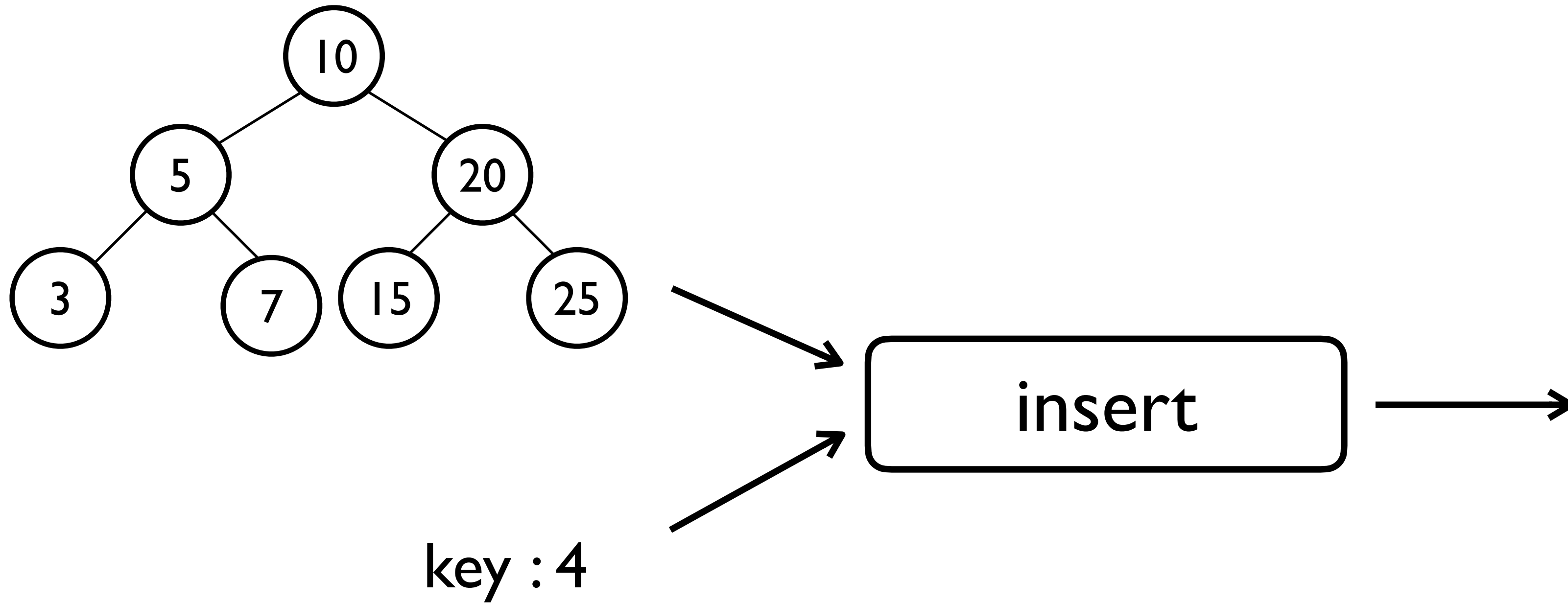
- 전위순회(VLR) :
- 후위순회(LRV) :
- 중위순회(LVR) :

이진 탐색 트리

- 이진 탐색 트리(BST, Binary Search Tree)는 다음과 같은 기능들 제공함
 - create(): 비어있는 이진 탐색 트리를 생성 후 반환함
 - insert(root, key): 이진 탐색 트리의 특성을 유지하면서 key를 키값으로 가지는 새로운 노드 n를 삽입함
 - findMin(root): 이진 탐색 트리에서 키값이 가장 작은 노드를 찾아서 반환함
 - delete(root, key): 이진 탐색 트리의 특성을 유지하면서 key를 키값으로 가지는 노드를 삭제함
 - search(root, key): 이진 탐색 트리에서 키값이 key인 노드를 찾아 반환함
 - traversal(root): 이진 탐색 트리를 구성하는 노드들의 키값들을 출력함
 - height(root): 이진 탐색 트리의 높이를 반환함

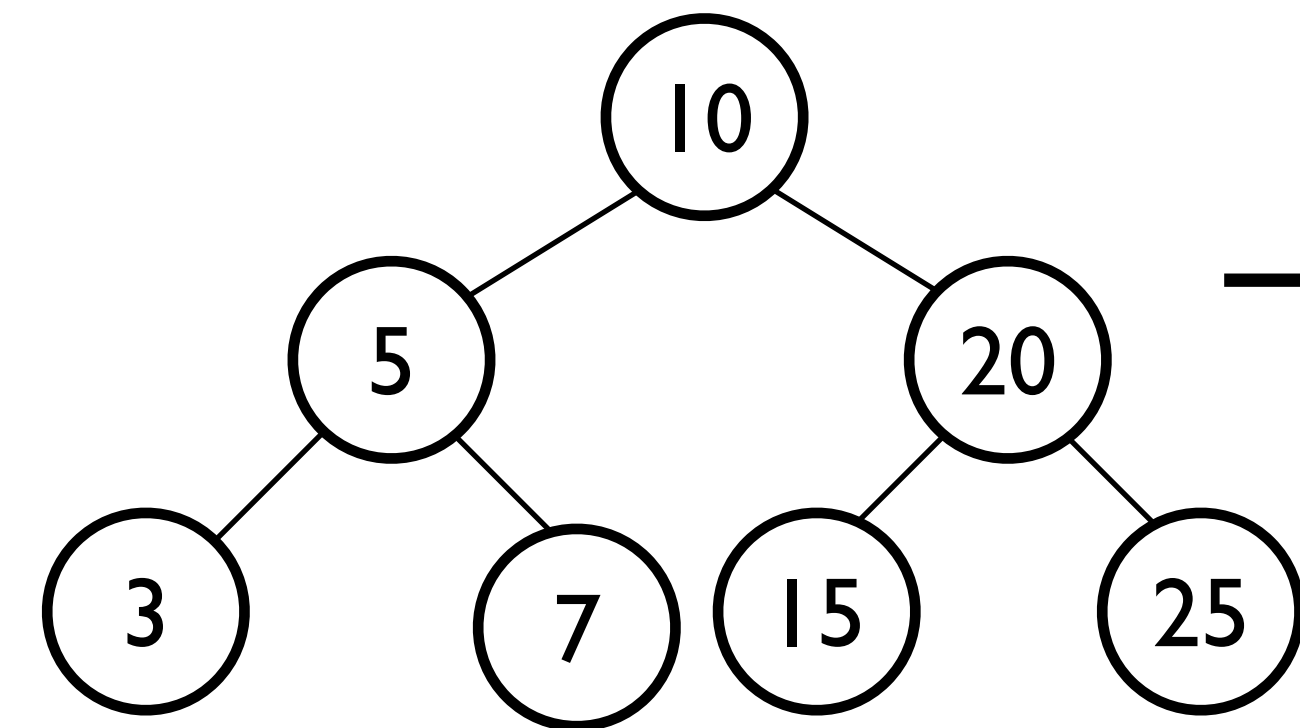
insert

- insert: 이진 탐색 트리의 특성을 유지하면서 주어진 키를 가지는 새로운 노드를 이진 탐색 트리에 삽입함



insert

- insert: 이진 탐색 트리의 특성을 유지하면서 주어진 키를 가지는 새로운 노드를 이진 탐색 트리에 삽입함

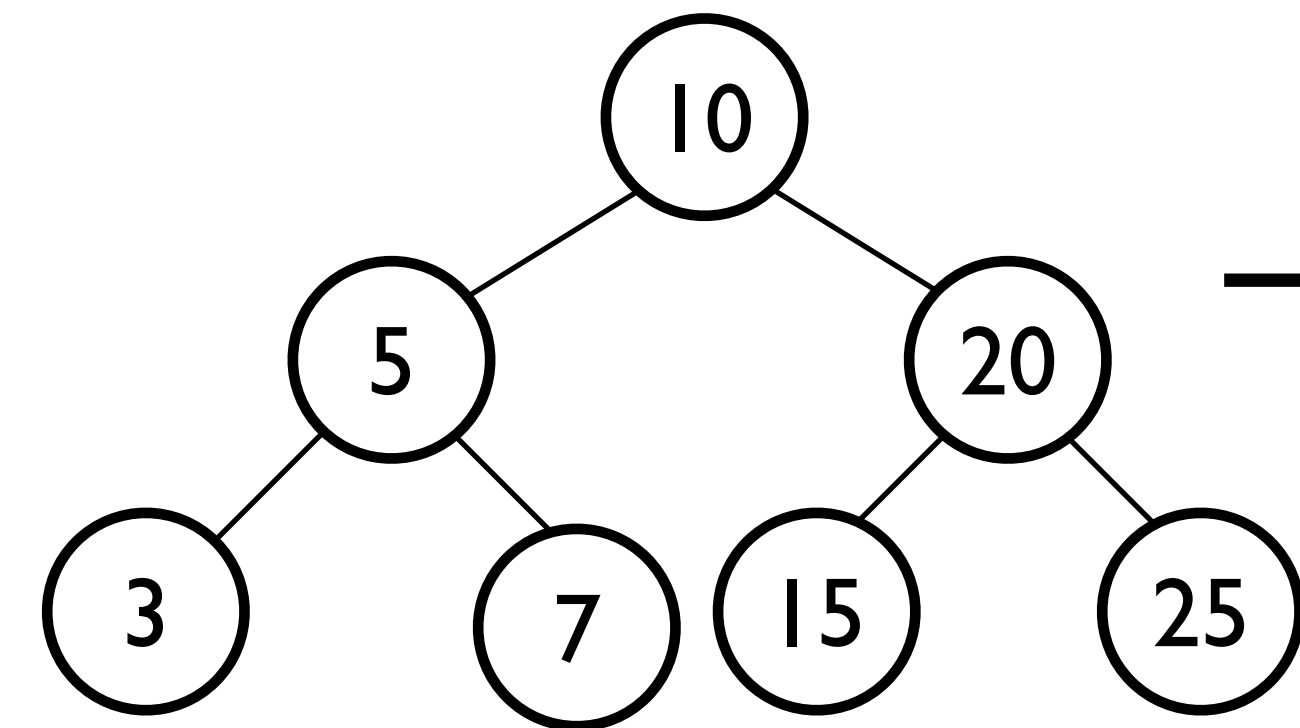


key : 4

```
procedure insert(root, key)
  if root = NULL then
    root ← allocateNode()
    root.key ← key
    root.left ← NULL
    root.right ← NULL
    return root
  end if
  if key < root.key then
    root.left ← insert(root.left, key)
  elif key > root.key then
    root.right ← insert(root.right, key)
  end if
  return root
end procedure
```

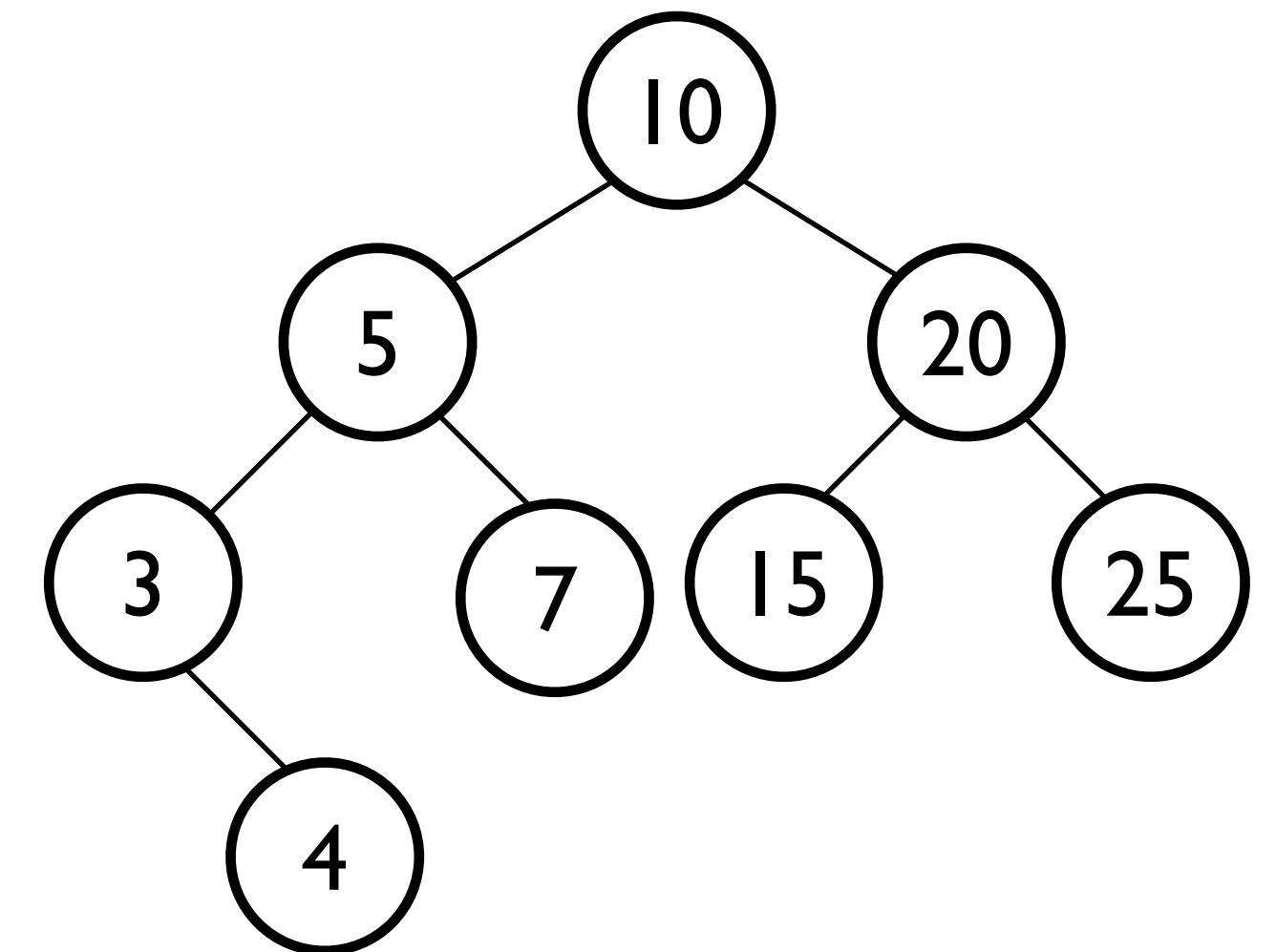
insert

- insert: 이진 탐색 트리의 특성을 유지하면서 주어진 키를 가지는 새로운 노드를 이진 탐색 트리에 삽입함



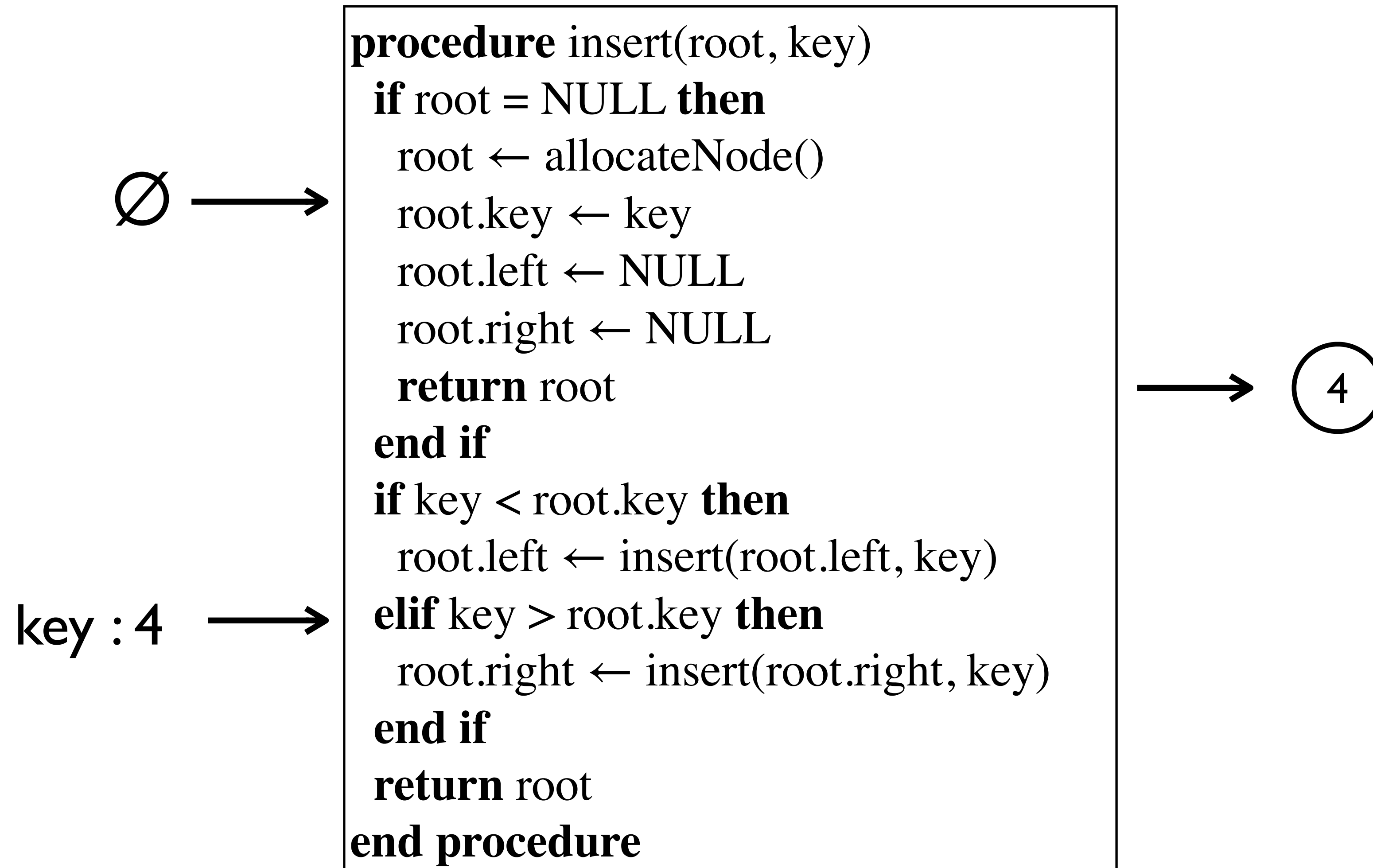
key : 4

```
procedure insert(root, key)
  if root = NULL then
    root ← allocateNode()
    root.key ← key
    root.left ← NULL
    root.right ← NULL
    return root
  end if
  if key < root.key then
    root.left ← insert(root.left, key)
  elif key > root.key then
    root.right ← insert(root.right, key)
  end if
  return root
end procedure
```



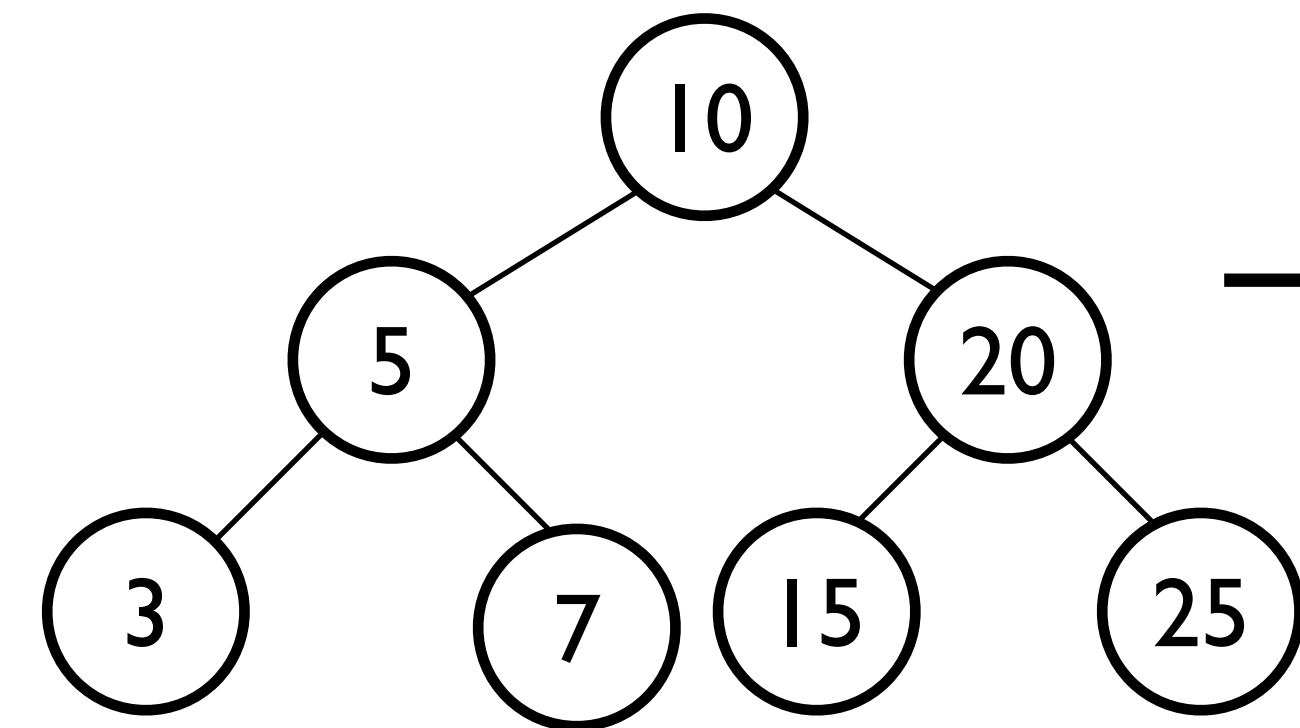
insert

- insert: 이진 탐색 트리의 특성을 유지하면서 주어진 키를 가지는 새로운 노드를 이진 탐색 트리에 삽입함



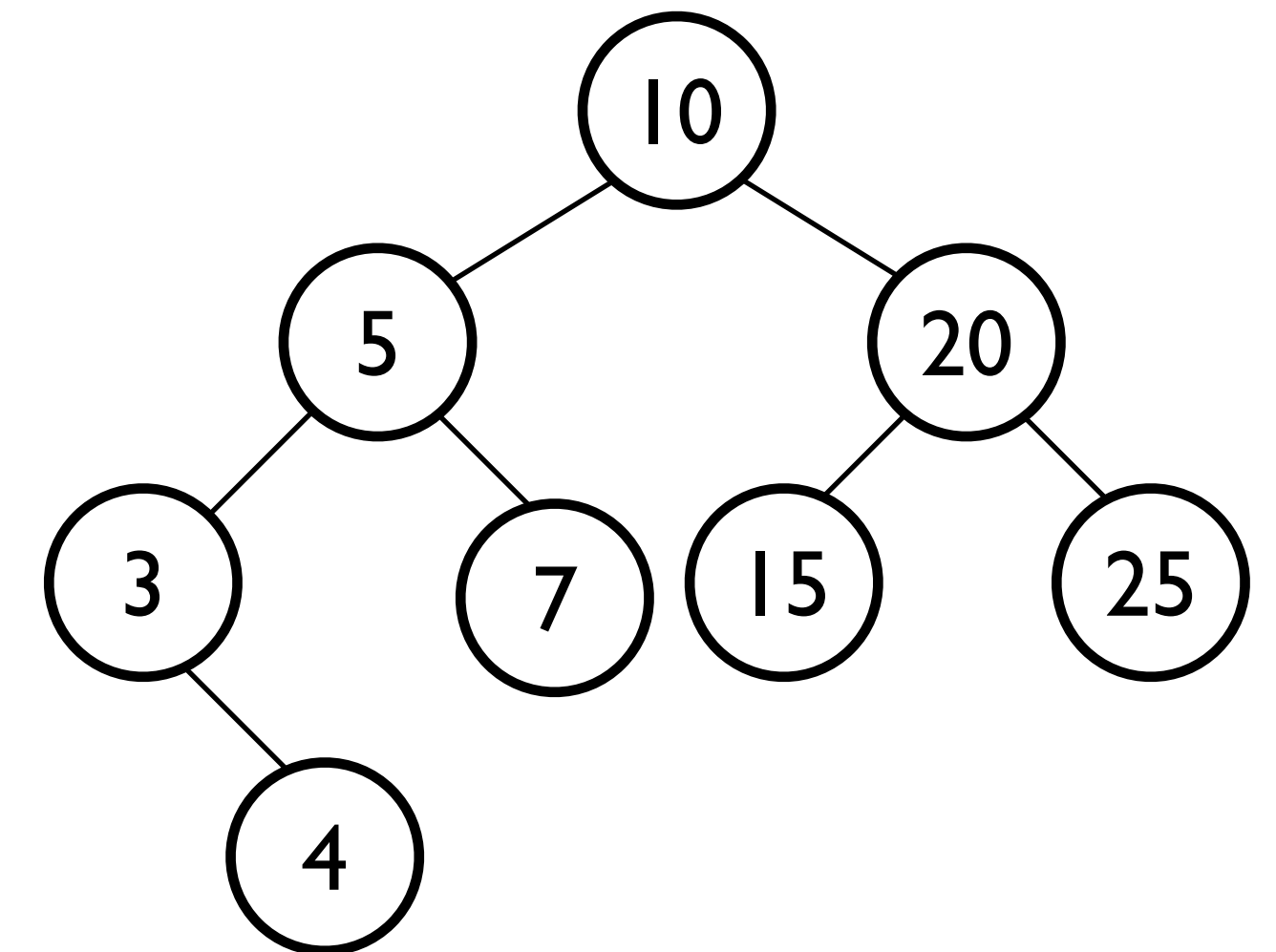
insert

- insert: 이진 탐색 트리의 특성을 유지하면서 주어진 키를 가지는 새로운 노드를 이진 탐색 트리에 삽입함



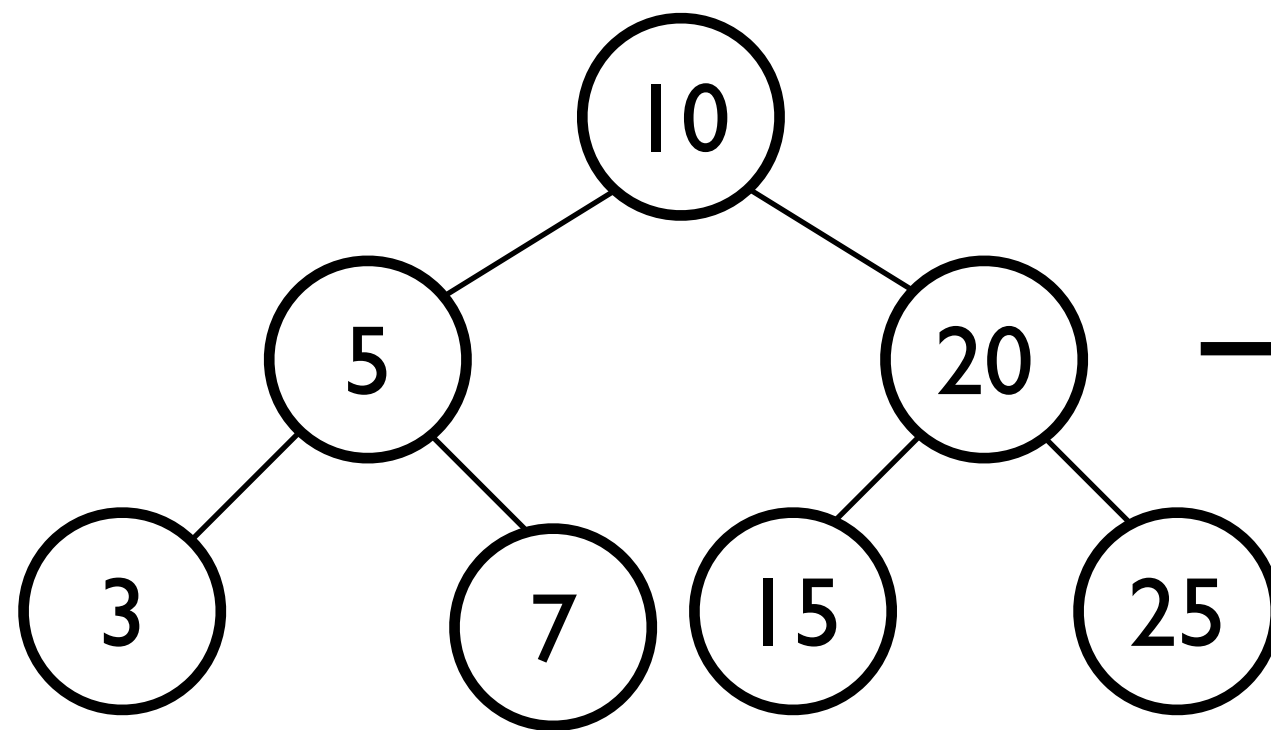
key : 4

```
procedure insert(root, key)
  if root = NULL then
    root ← allocateNode()
    root.key ← key
    root.left ← NULL
    root.right ← NULL
    return root
  end if
  if key < root.key then
    root.left ← insert(root.left, key)
  elif key > root.key then
    root.right ← insert(root.right, key)
  end if
  return root
end procedure
```

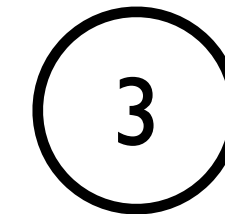


findMin

- findMin: 이진 탐색 트리에서 키값이 가장 작은 노드를 찾아서 반환함

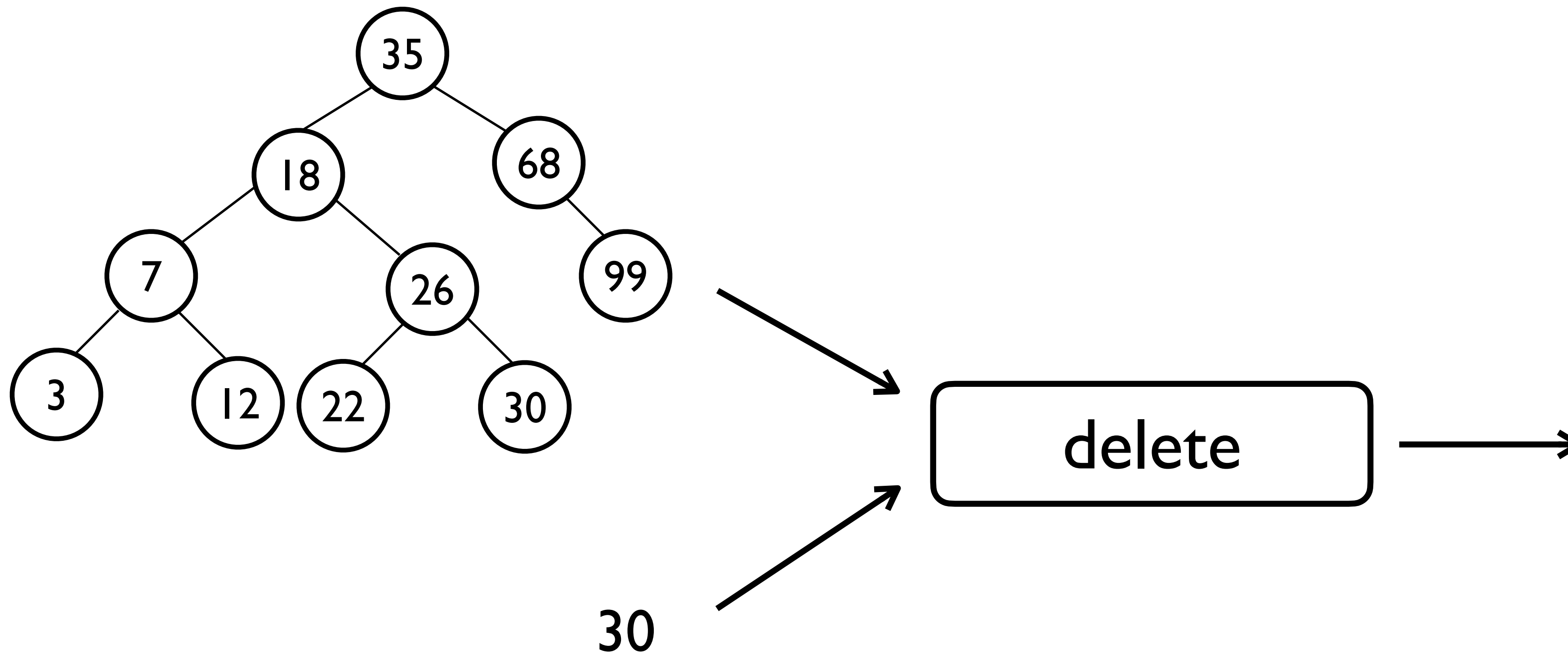


```
procedure findMin(root)
if root = NULL then
  return NULL
end if
node ← root
while node.left ≠ NULL do
  node ← node.left
end while
return node
end procedure
```



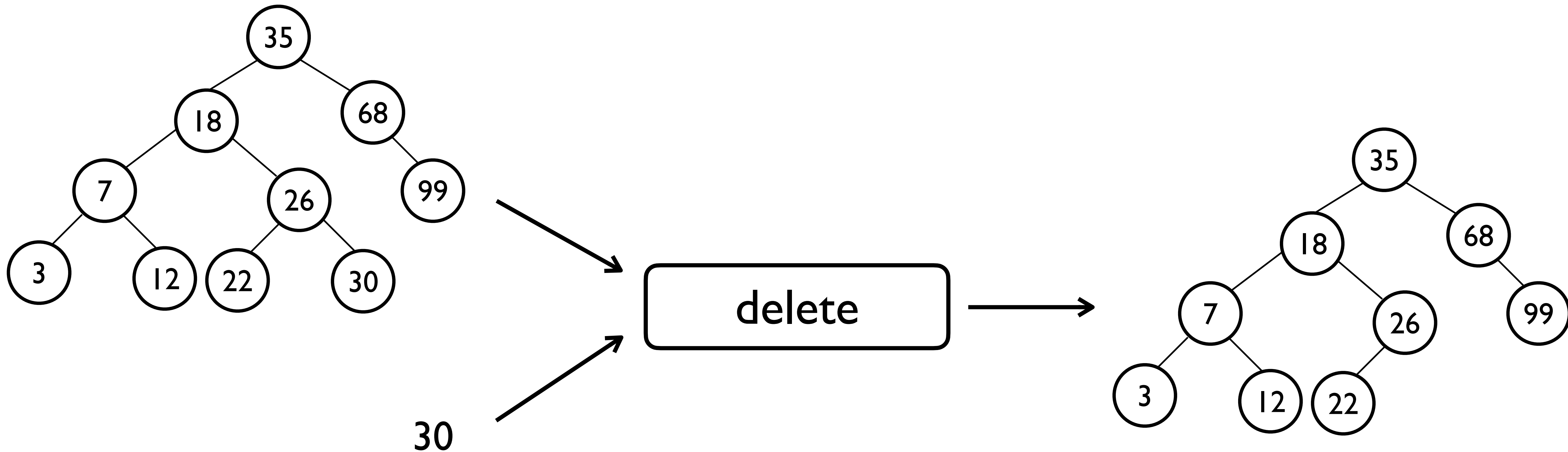
delete

- delete: 이진 탐색 트리의 특성을 유지하면서 주어진 키값을 가진 노드를 이진 탐색 트리에서 삭제함
 - Case 1: 삭제하려는 노드가 리프 노드일 경우



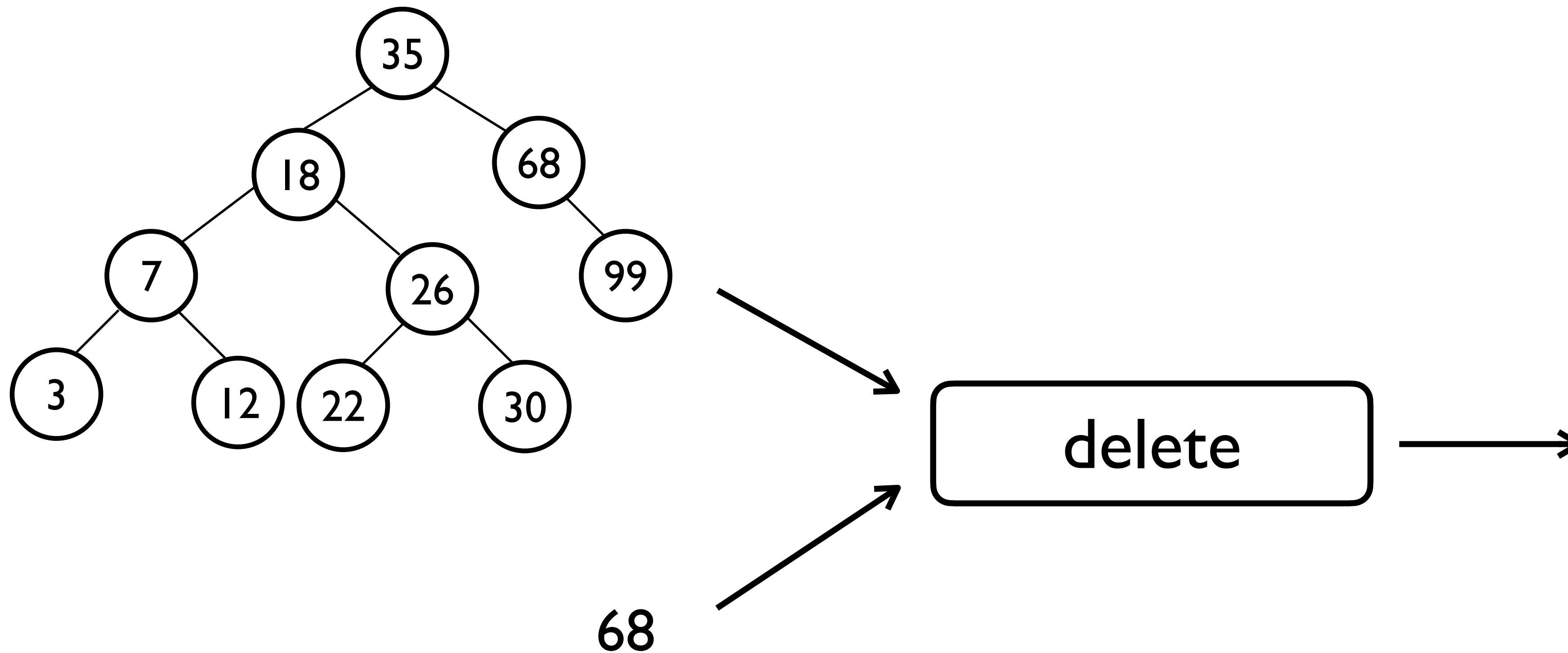
delete

- delete: 이진 탐색 트리의 특성을 유지하면서 주어진 키값을 가진 노드를 이진 탐색 트리에서 삭제함
 - Case 1: 삭제하려는 노드가 리프 노드일 경우



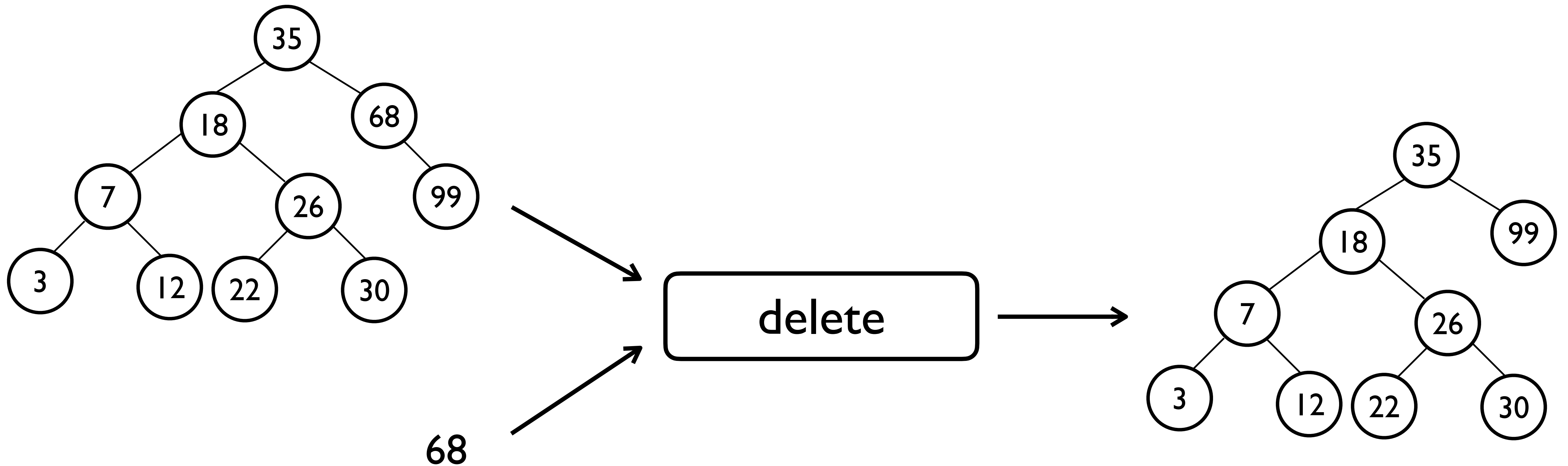
delete

- delete: 이진 탐색 트리의 특성을 유지하면서 주어진 키값을 가진 노드를 이진 탐색 트리에서 삭제함
 - Case 2: 삭제하려는 노드가 하나의 서브트리만 가지고 있는 경우



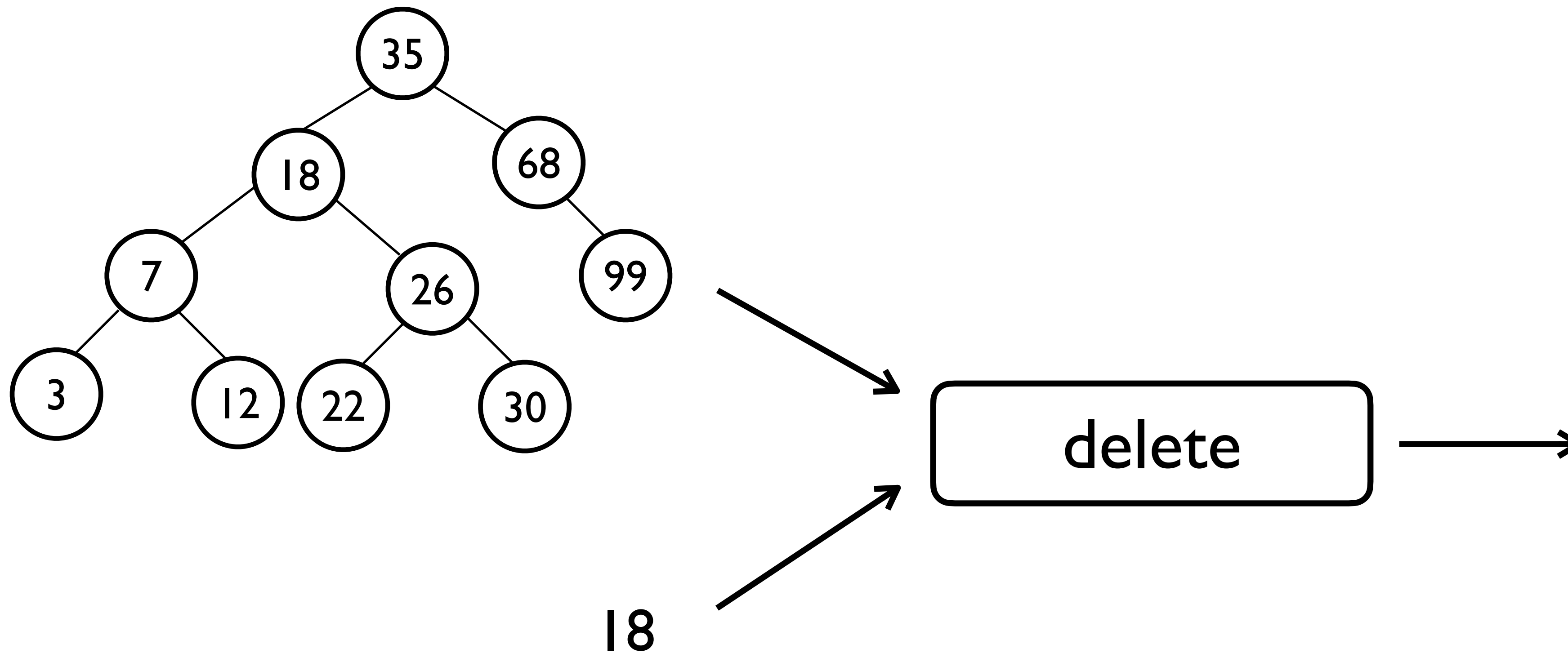
delete

- delete: 이진 탐색 트리의 특성을 유지하면서 주어진 키값을 가진 노드를 이진 탐색 트리에서 삭제함
 - Case 2: 삭제하려는 노드가 하나의 서브트리만 가지고 있는 경우



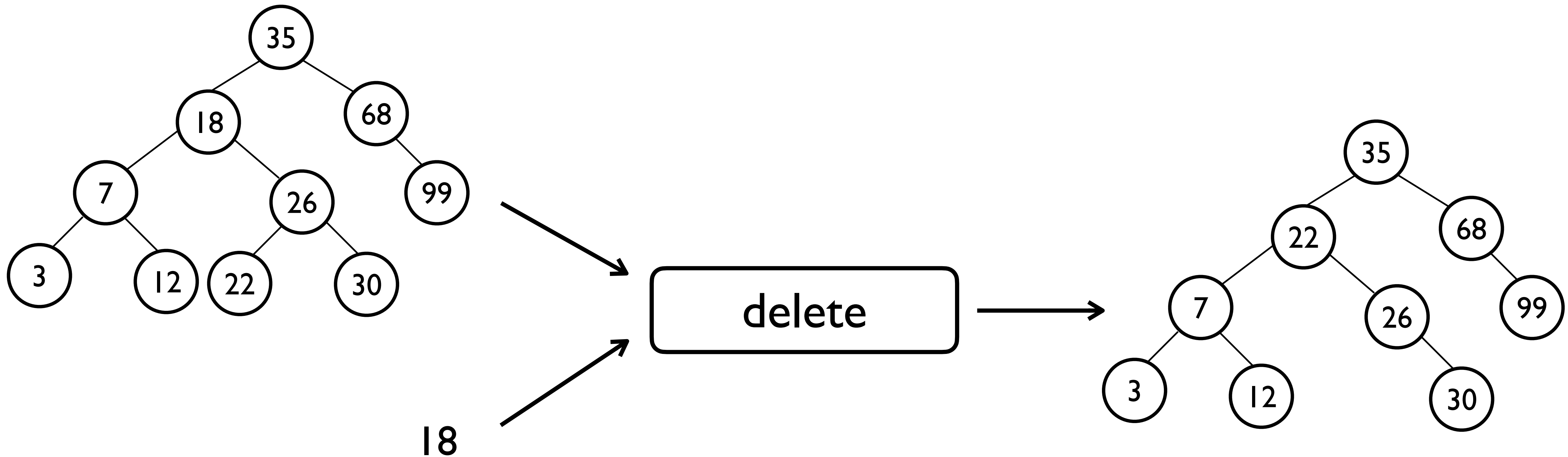
delete

- delete: 이진 탐색 트리의 특성을 유지하면서 주어진 키값을 가진 노드를 이진 탐색 트리에서 삭제함
 - Case 3: 삭제하려는 노드가 두개의 서브트리를 가지고 있는 경우



delete

- delete: 이진 탐색 트리의 특성을 유지하면서 주어진 키값을 가진 노드를 이진 탐색 트리에서 삭제함
 - Case 3: 삭제하려는 노드가 두개의 서브트리를 가지고 있는 경우

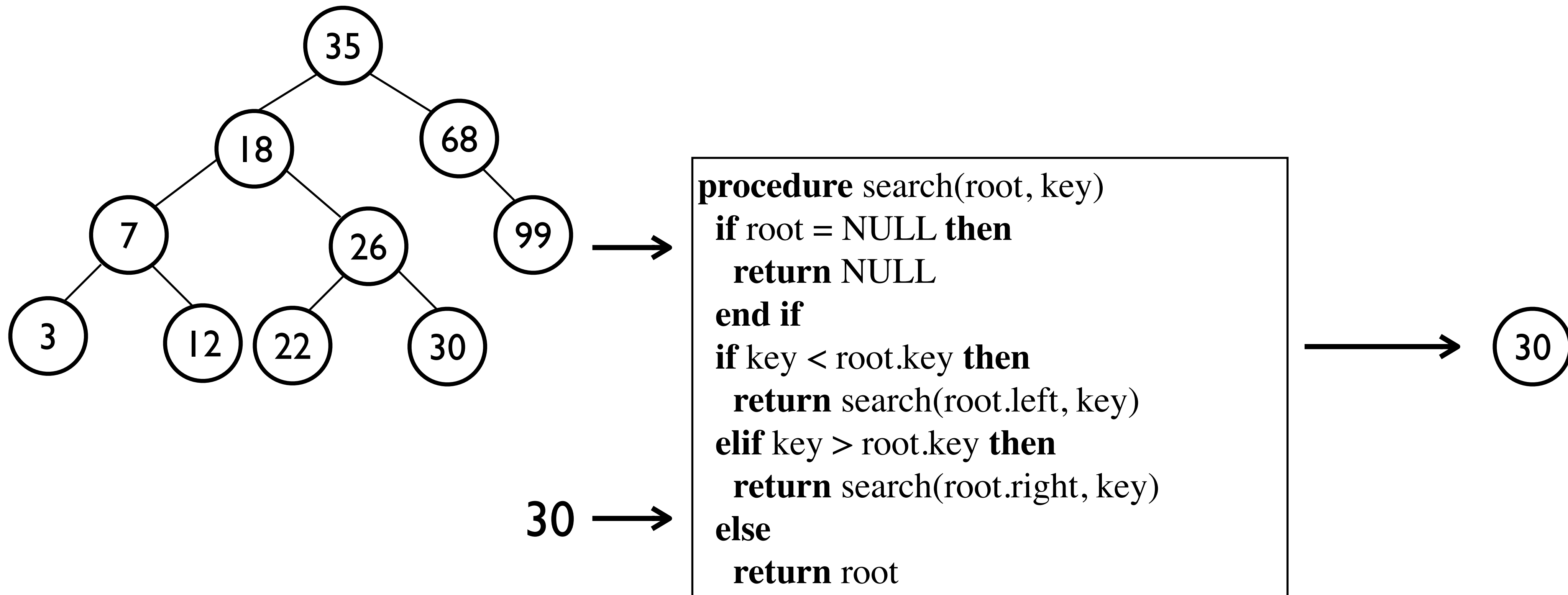


delete

```
procedure delete(root, key)
  if root = NULL then
    return NULL
  end if
  if key < root.key then
    root.left ← deleteNode(root.left, key)
  elif key > root.key then
    root.right ← deleteNode(root.right, key)
  else
    if root.left = NULL then
      node ← root.right
      free(root)
      return node
    elif root.right = NULL then
      node ← root.left
      free(right)
      return node
    end if
    node ← findMin(root.right)
    root.key ← node.key
    root.right ← delete(root.right, node.key)
  end if
  return root
end procedure
```

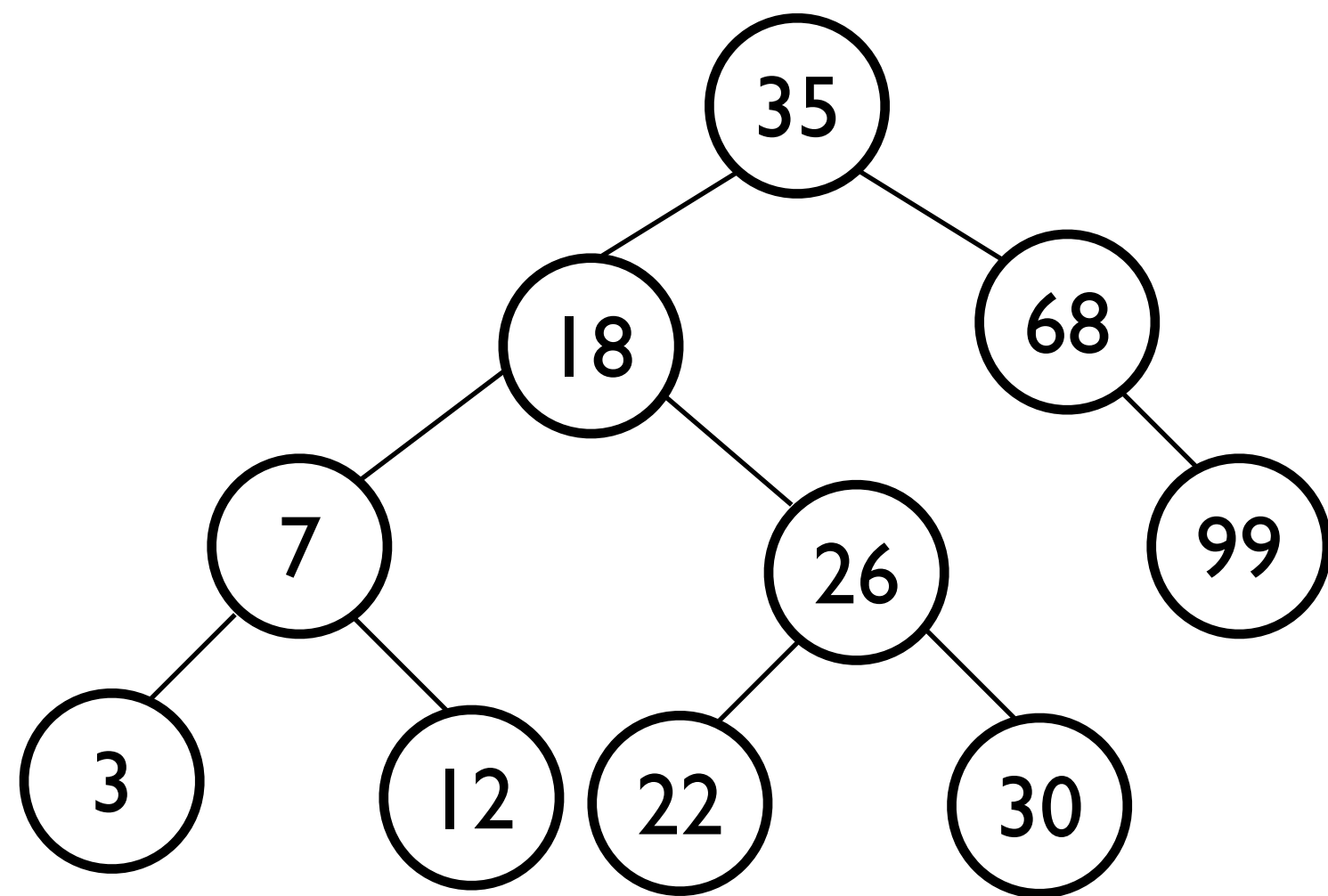
search

- search: 이진 탐색 트리에서 키값이 key인 노드를 찾아 반환함



traversal

- traversal: 이진 탐색 트리를 구성하는 노드들의 key값들을 출력함

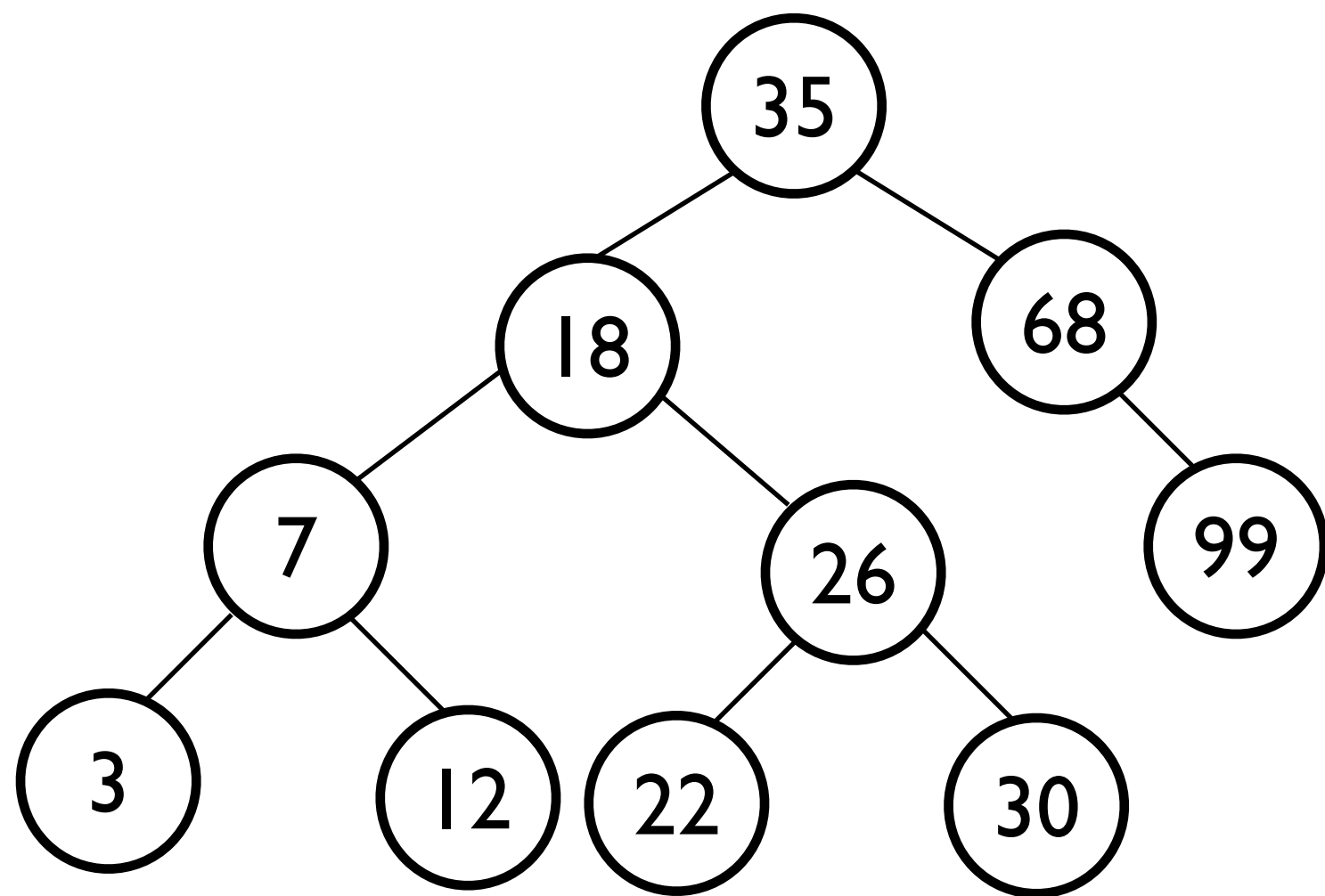


```
procedure traversal(root)
  if root ≠ NULL then
    traversal(root.left)
    print(root.key)
    traversal(root.right)
  end if
```

```
Keys in BST:
3, 7, 12, 18, 22, 26,
30, 35, 68, 99
```


height

- height: 이진 탐색 트리의 높이를 반환함



```
procedure height(root)
  if root = NULL then
    return 0
  end if

  leftHeight ← height(root.left)
  rightHeight ← height(root.right)

  if leftHeight > rightHeight then
    return leftHeight + 1
  else
    return rightHeight + 1
  end if
```



4

이진 탐색 트리의 구현

- 이진 탐색 트리(BST, Binary Search Tree)는 다음과 같은 연산을 제공함 (이진 탐색 트리의 추상 자료형)
 - `Node* create()`:
 - 비어있는 이진트리를 생성 후 반환함
 - `Node* insert(Node* root, int key)`:
 - 이진 탐색 트리의 특성을 유지하면서 `key`를 키값으로 가지는 노드를 이진 탐색 트리에 삽입 후 루트 노드를 반환
 - `Node* findMin(Node* node)`:
 - 이진 탐색 트리에서 `key`값이 최소인 노드를 반환함
 - `Node* deleteNode(Node* root, int key)`:
 - 이진 탐색 트리의 특성을 유지하면서 노드 `key`를 키값으로 가지는 노드를 이진 탐색 트리에서 삭제 후 루트 노드를 반환
 - `Node* search(Node* root, int key)`:
 - 이진 탐색 트리에서 키값이 `key`인 노드를 찾아 반환함
 - `void traversal(Node* root)`:
 - 이진 탐색 트리를 구성하는 노드들의 키값들을 오름차순으로 출력함
 - `int height(Node* root)`:
 - 이진 탐색 트리의 높이를 반환함

Example

```
#include <stdio.h>
#include "BST.h"

int main() {
    Node* root = create();
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);
    root = insert(root, 90);

    printf("키값들을 순회: ");
    traversal(root);
    printf("\n");
    printf("Tree의 높이 : %d\n", height(root));

    root = deleteNode(root, 50);
    root = deleteNode(root, 60);
    printf("Tree의 높이 : %d\n", height(root));
    root = deleteNode(root, 70);
    root = deleteNode(root, 80);
    root = deleteNode(root, 90);
    root = deleteNode(root, 30);
    root = deleteNode(root, 40);
    root = deleteNode(root, 20);
    printf("Tree의 높이 : %d\n", height(root));
    return 0;
}
```

이진 탐색 트리의 구현

- 이진트리의 노드는 다음과 같은 정보를 가짐

```
typedef struct Node {  
    int key;  
    struct Node* left;  
    struct Node* right;  
} Node;
```

- create: 비어있는 이진 탐색 트리를 생성 후 반환함

```
procedure create()  
    return NULL  
end procedure
```

```
Node* create(){  
    return NULL;  
}
```


이진 탐색 트리의 구현

- search: 이진 탐색 트리에서 주어진 키값을 가지는 노드를 찾아 반환함

```
procedure search(root, key)
  if root = NULL then
    return NULL
  end if
  if key < root.key then
    return search(root.left, key)
  elif key > root.key then
    return search(root.right, key)
  else
    return root
```

```
Node* search(Node* root, int key) {
```

이진 탐색 트리의 구현

- deleteNode: 이진 탐색 트리의 특성을 유지하면서 노드 key를 키값으로 가지는 노드를 이진 탐색 트리에서 삭제함

```
procedure deleteNode(root, key)
  if root = NULL then
    return NULL
  end if
  if key < root.key then
    root.left ← deleteNode(root.left, key)
  elif key > root.key then
    root.right ← deleteNode(root.right, key)
  else
    if root.left = NULL then
      node ← root.right
      free(root)
      return node
    elif root.right = NULL then
      node ← root.left
      free(root)
      return node
    end if
    node ← findMin(root.right)
    root.key ← node.key
    root.right ← deleteNode(root.right, node.key)
  end if
  return root
end procedure
```

```
Node* deleteNode(Node* root, int key) {
```


이진 탐색 트리의 구현

- traversal: 이진 탐색 트리를 구성하는 노드들의 키값들을 오름차순으로 출력함

```
procedure traversal(root)
  if root  $\neq$  NULL then
    traversal(root.left)
    print(root.key)
    traversal(root.right)
  end if
```

```
void traversal(Node* root) {
  

}
```


마무리

- 이진 탐색 트리(BST, Binary Search Tree)
 - 이진 탐색 트리(BST, Binary Search Tree)는 이진트리 기반의 탐색을 위한 자료구조임
 - 이진 탐색 트리는 다음과 같이 (재귀적으로) 정의됨

Definition

- (1) 모든 노드는 유일한 키(key)를 갖는다.
- (2) 왼쪽 서브트리의 키들은 루트의 키보다 작다.
- (3) 오른쪽 서브트리의 키들은 루트의 키보다 크다.
- (4) 왼쪽과 오른쪽 서브트리도 이진 탐색 트리이다.

