COSE213: Data Structure

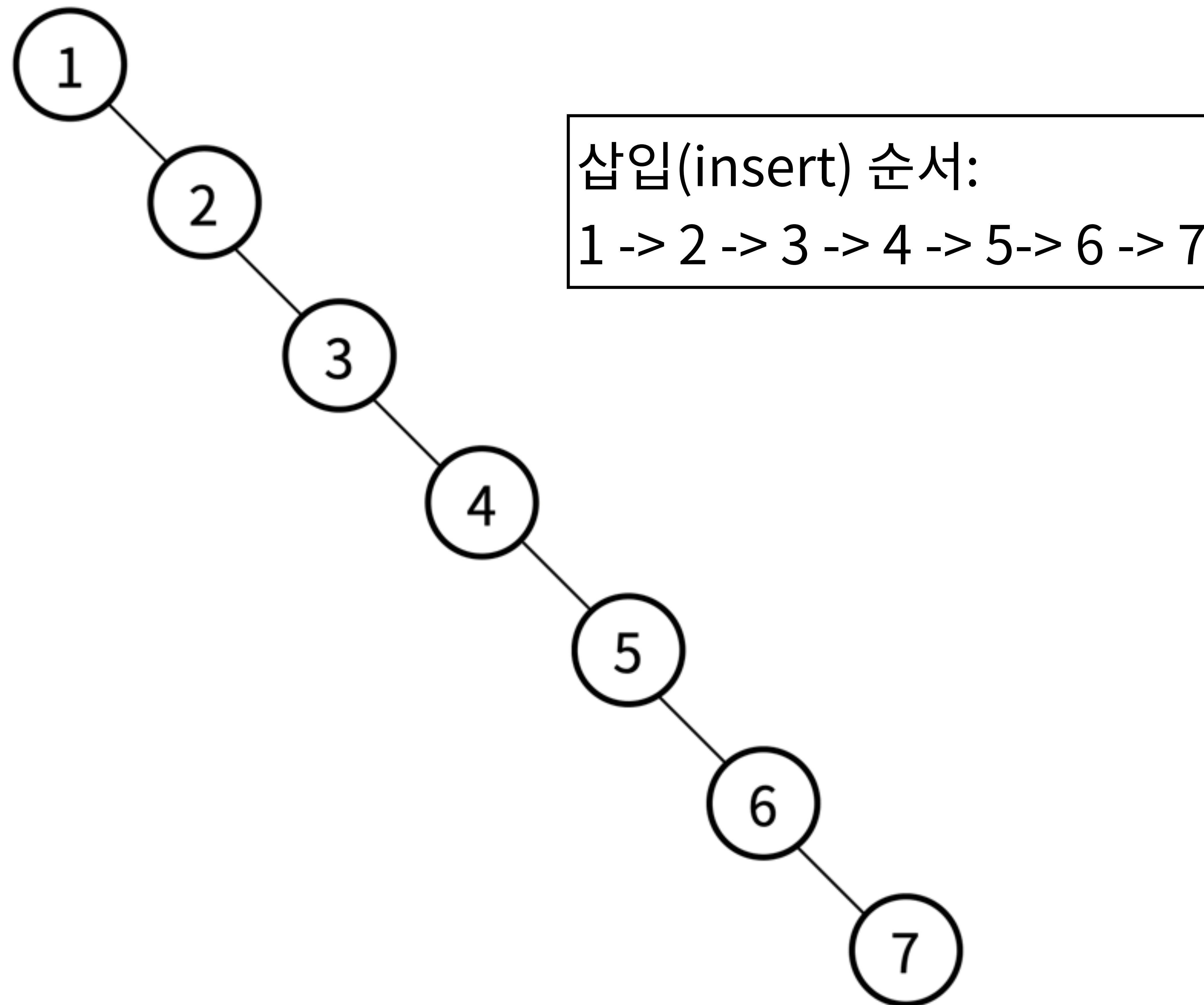
Lecture 9 - AVL 트리 (AVL Tree)

Minseok Jeon

2024 Fall

일반 이진 탐색 트리의 문제점

- 아래와 같이 이진 탐색 트리가 균형을 이루지 않는 경우 탐색 알고리즘들의 시간 복잡도: $O(n)$



이진 탐색 트리

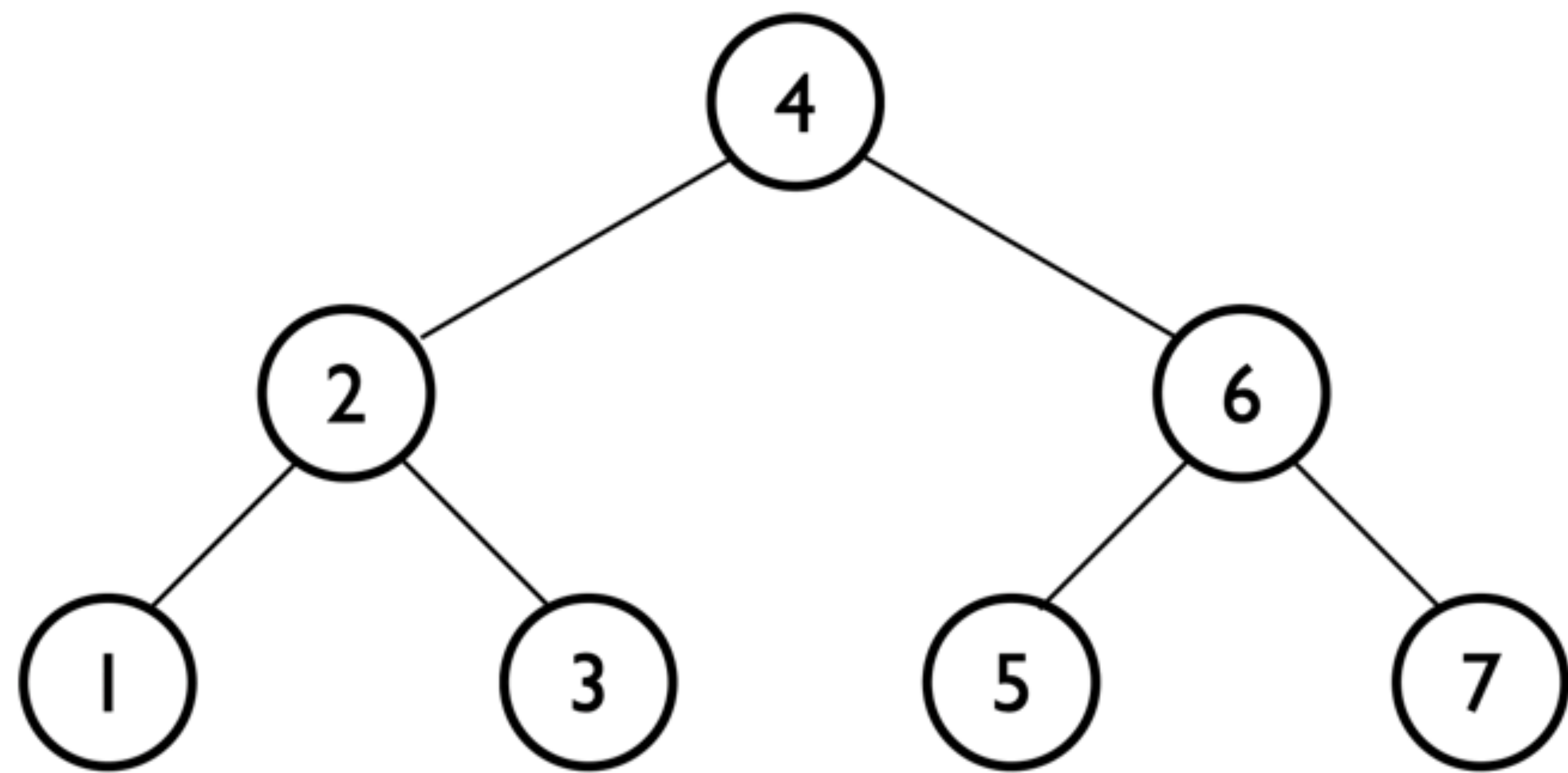
- 탐색 알고리즘

```
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
```

해결책: AVL 트리

- AVL 트리: 항상 균형을 이루는 이진 탐색 트리
  - 탐색 알고리즘의 시간 복잡도 : $O(\log n)$

삽입(insert) 순서:
1 -> 2 -> 3 -> 4 -> 5-> 6 -> 7



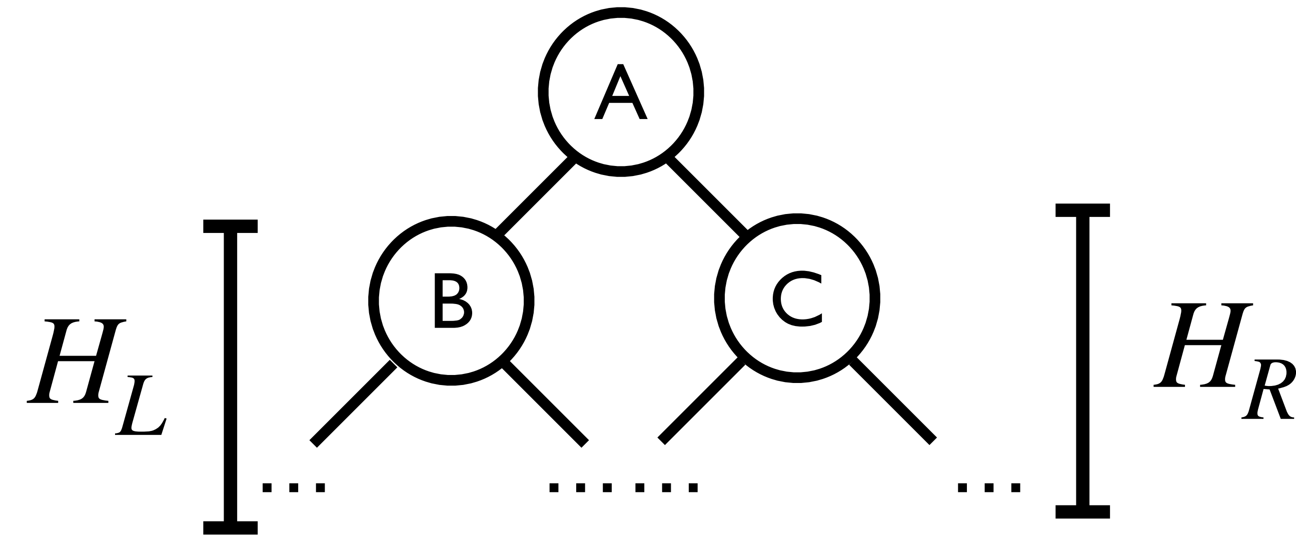
AVL 탐색 트리

- 탐색 알고리즘

```
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
```

이진트리 (Binary Tree)의 균형

- 균형(Balance)
  - 이진 트리의 왼쪽 서브트리와 오른쪽 서브트리의 높이를 각각 H_L과 H_R라 하자



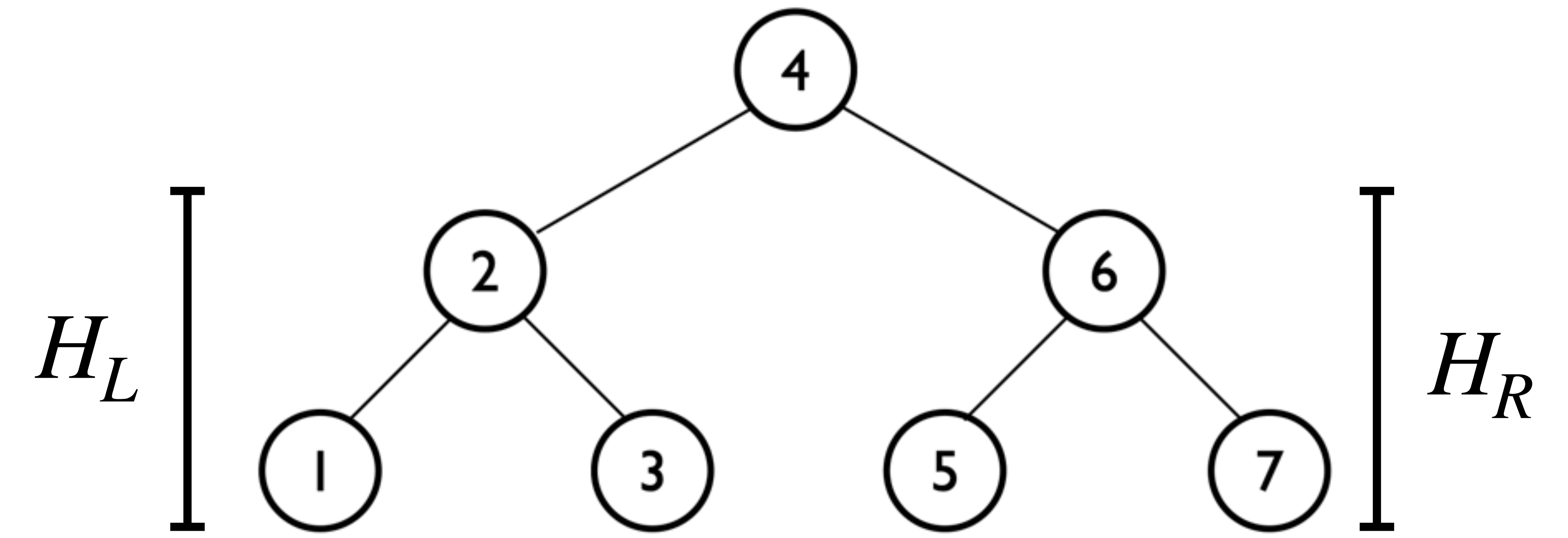
  - 이 때 균형 인수 (balance factor) B는 아래와 같이 정의됨

$$B = H_L - H_R$$

  - B의 값이 -1, 0, 1이면 균형을 이룬 트리라 함 (balanced tree)

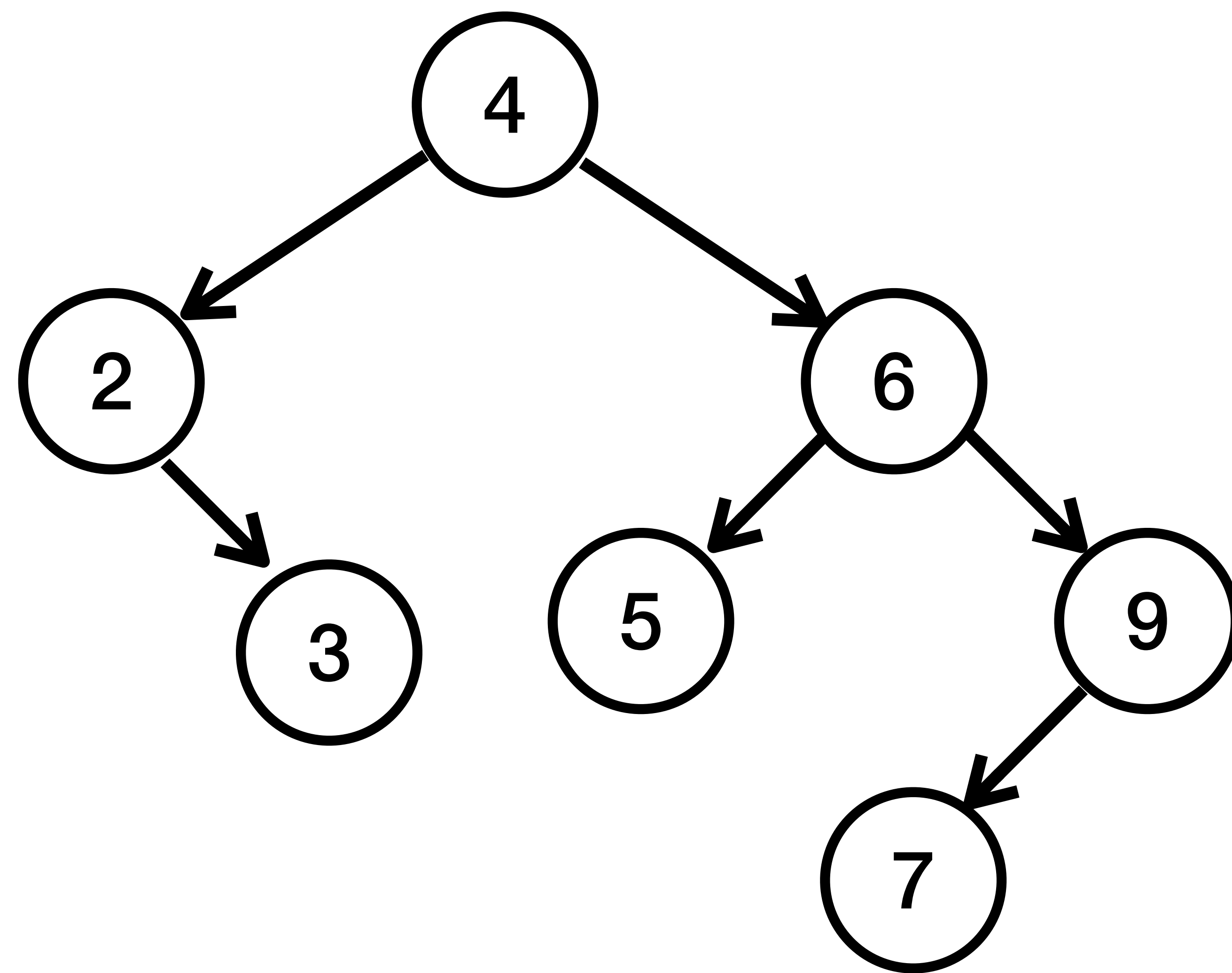
AVL 트리

- AVL 트리: 항상 균형을 이루는 이진 탐색 트리
- AVL 트리가 만족해야 할 성질
    - 균형인수 B $(H_L - H_R)$가 -1, 0, 또는 1
        - $B = 1$: 왼쪽 서브트리의 높이가 1 더 큼
        - $B = 0$: 두 서브트리의 높이가 같음
        - $B = -1$: 오른쪽 서브트리의 높이가 1 더 큼
- AVL 탐색 트리의 **서브트리도 AVL 트리**임

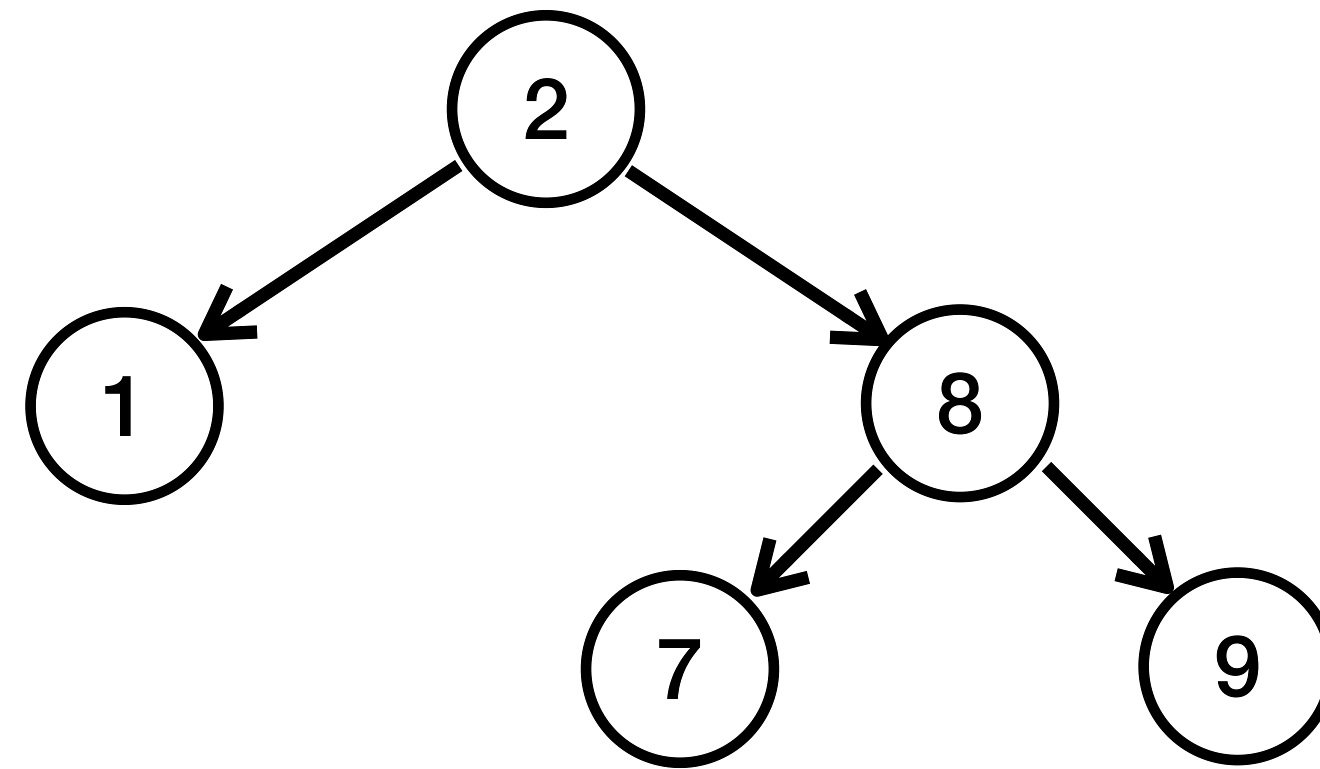


Example

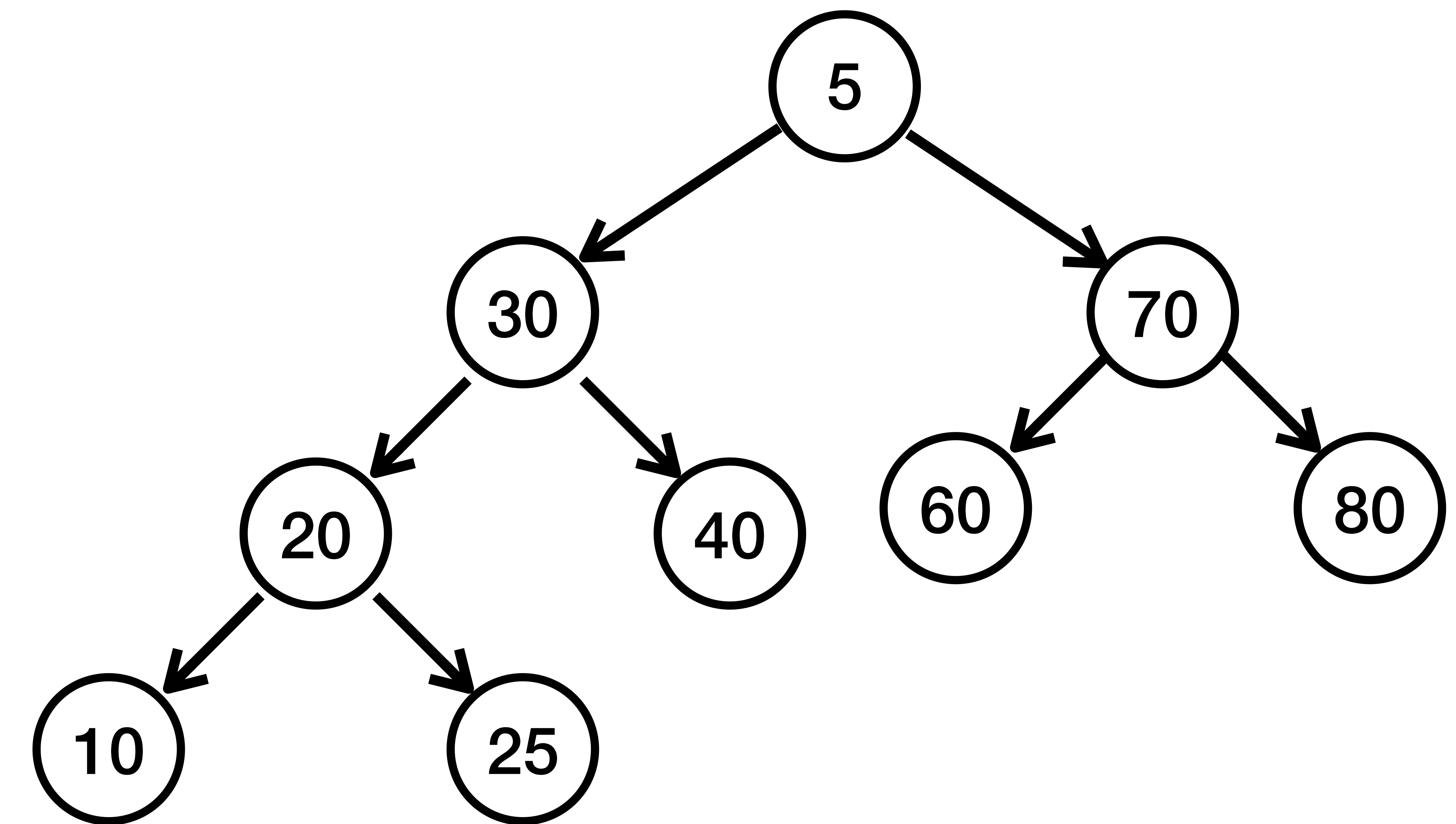
- Example: 다음 이진트리들은 AVL트리인가?



(a)



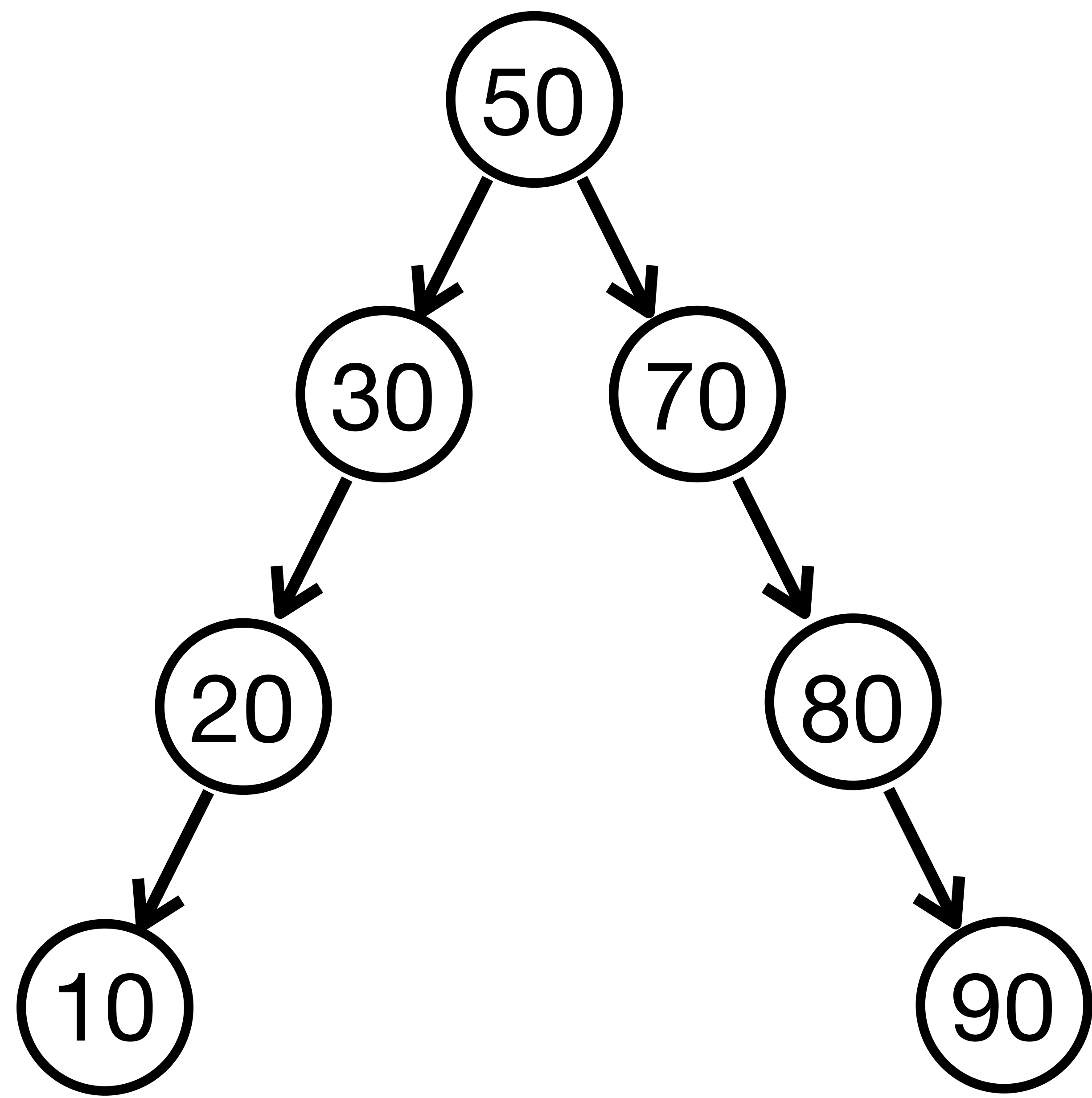
(b)



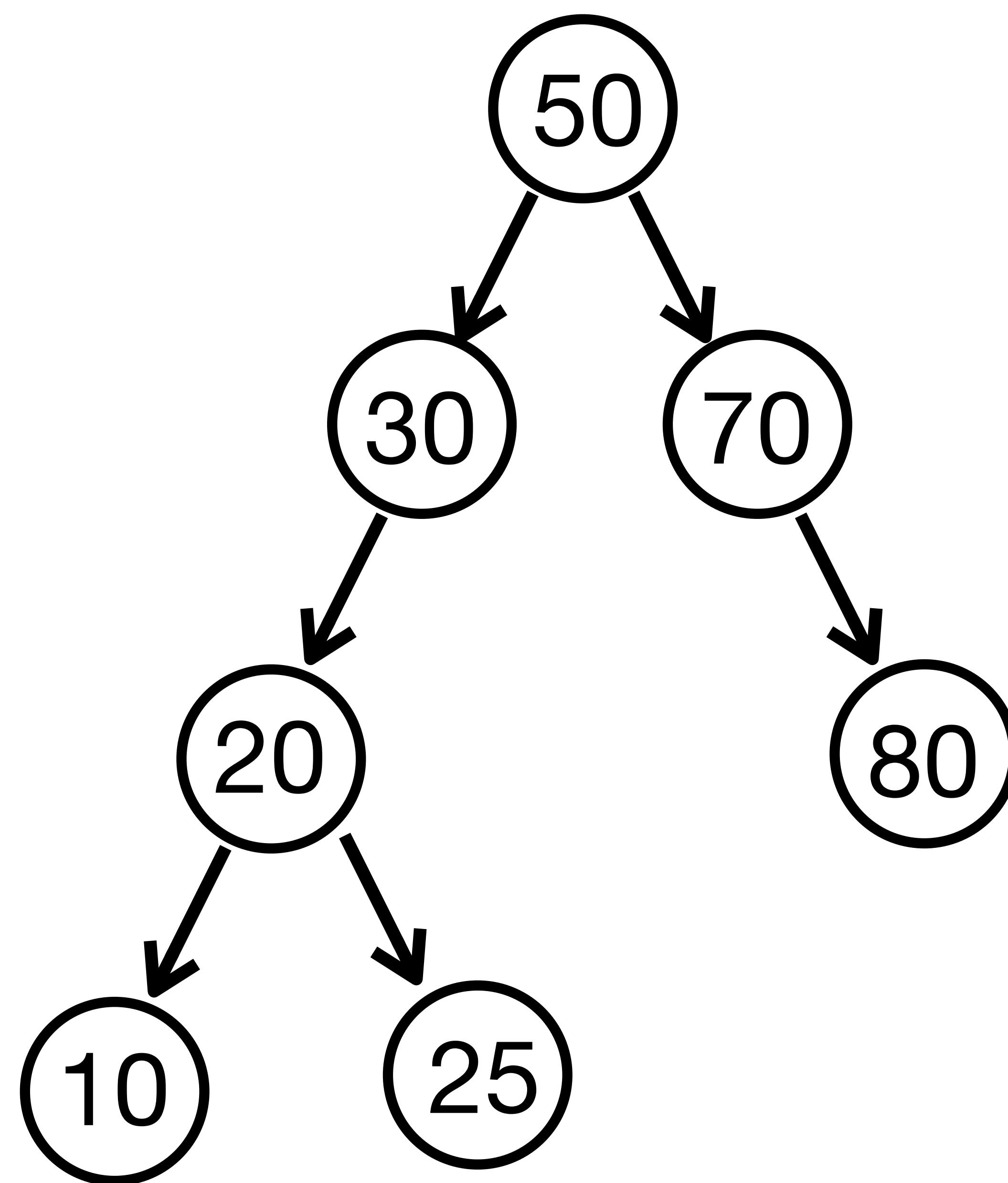
(c)

Example

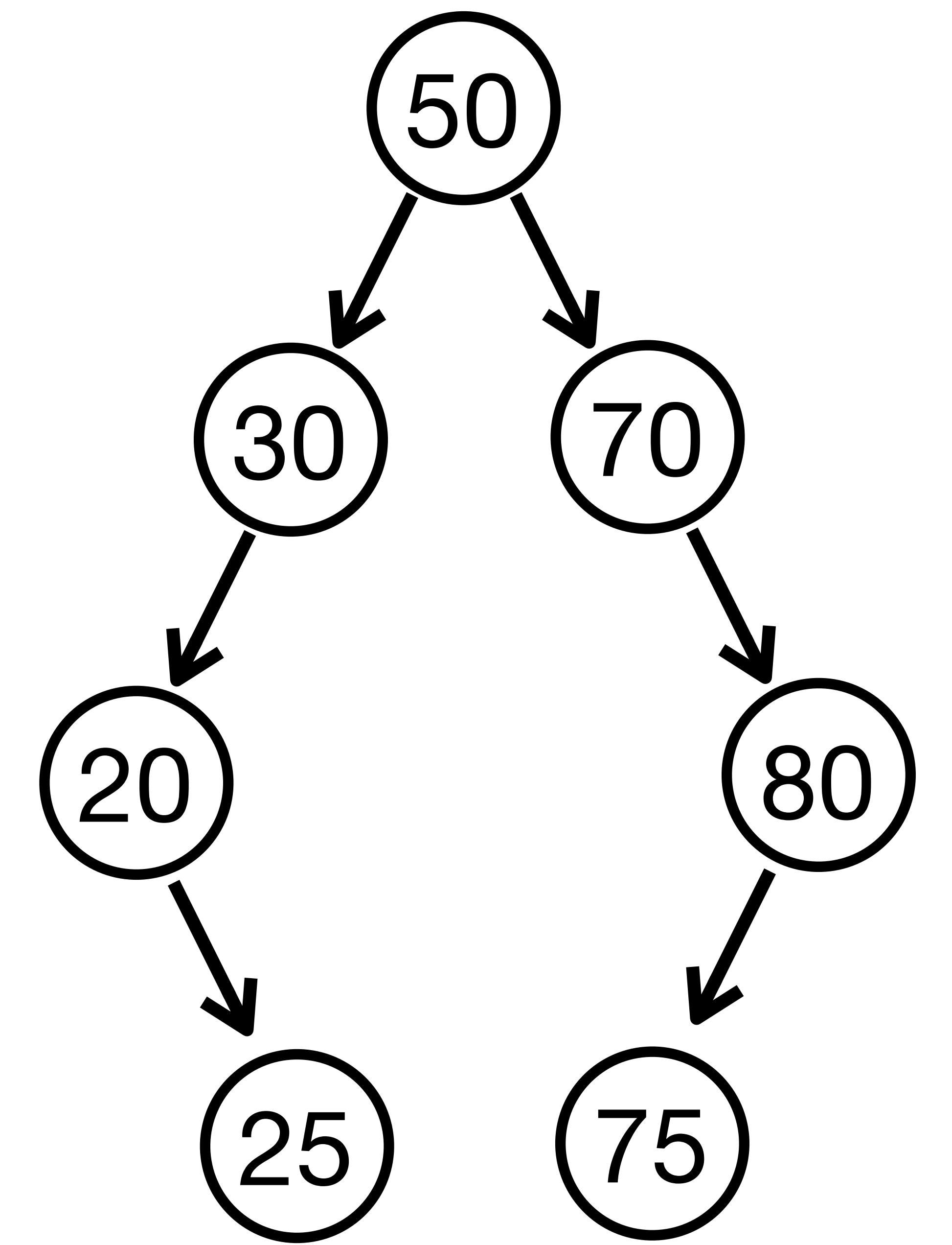
- Example: 다음 이진트리들은 AVL트리인가?



(a)



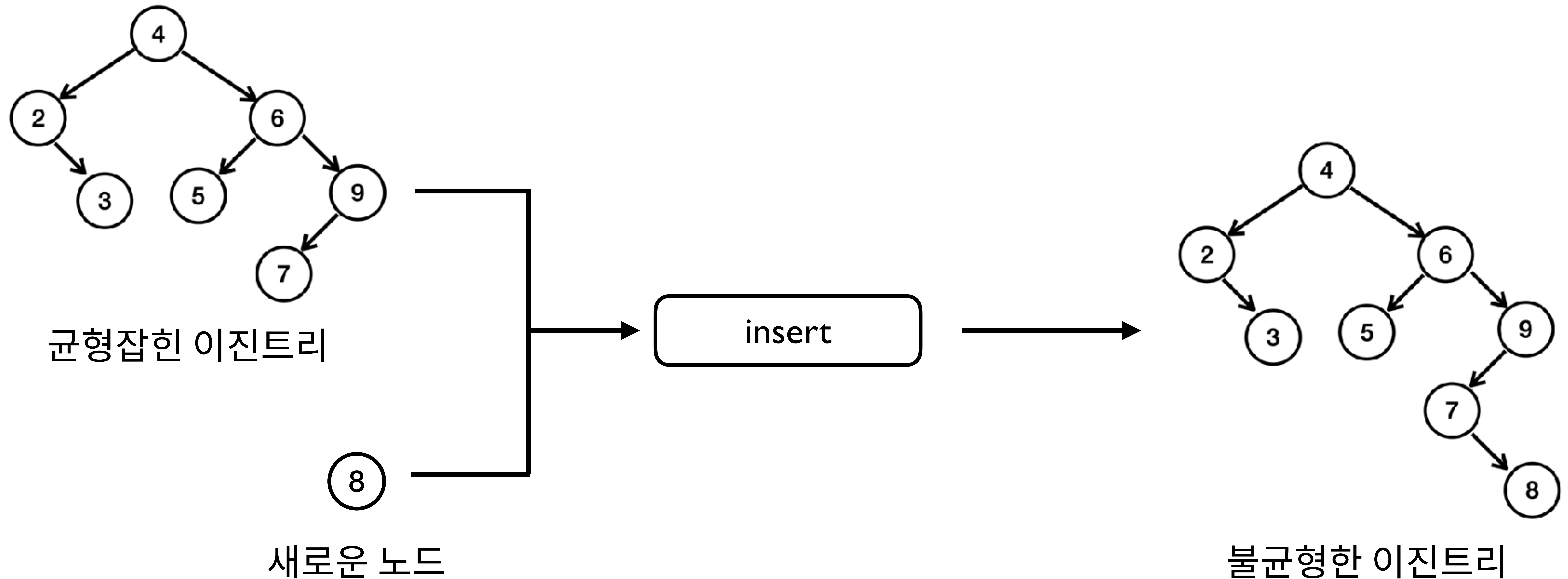
(b)



(c)

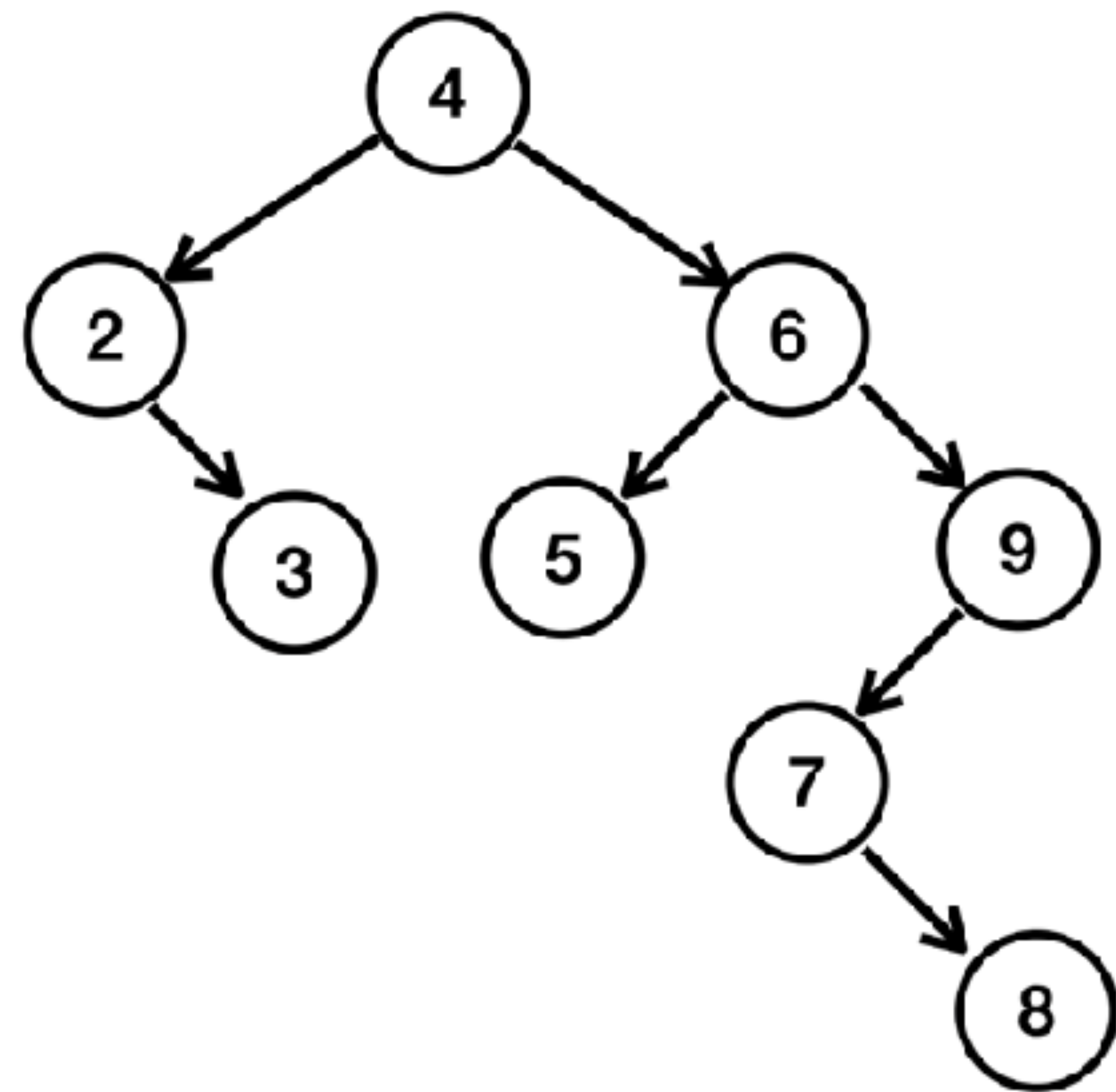
이진 탐색 트리에서의 삽입(insert)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음



트리 균형 잡기 (Balancing)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음
  - 균형이 깨질 경우 노드 회전(rotation)을 통해 균형을 잡아야 함



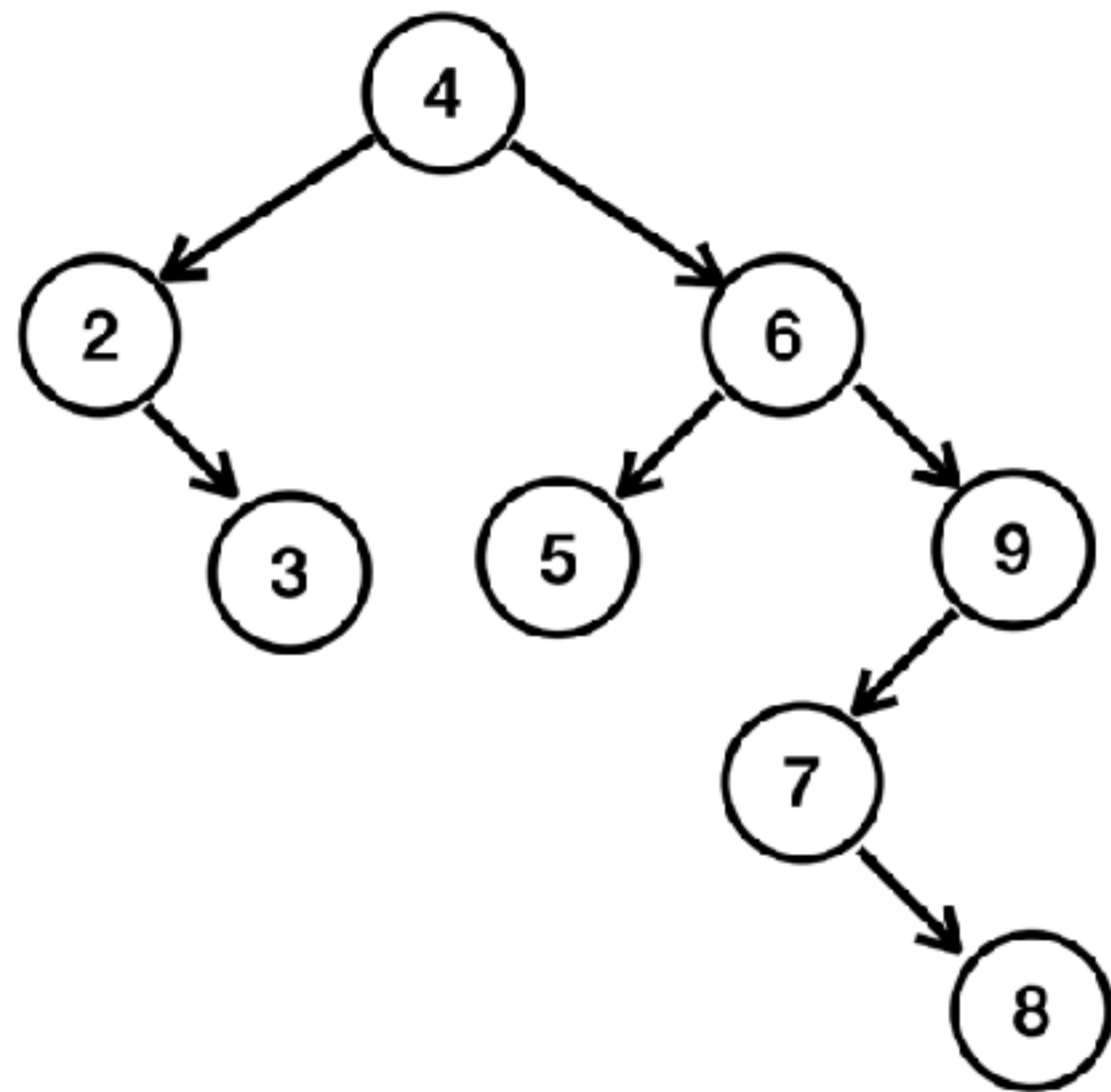
balancing

불균형한 이진트리

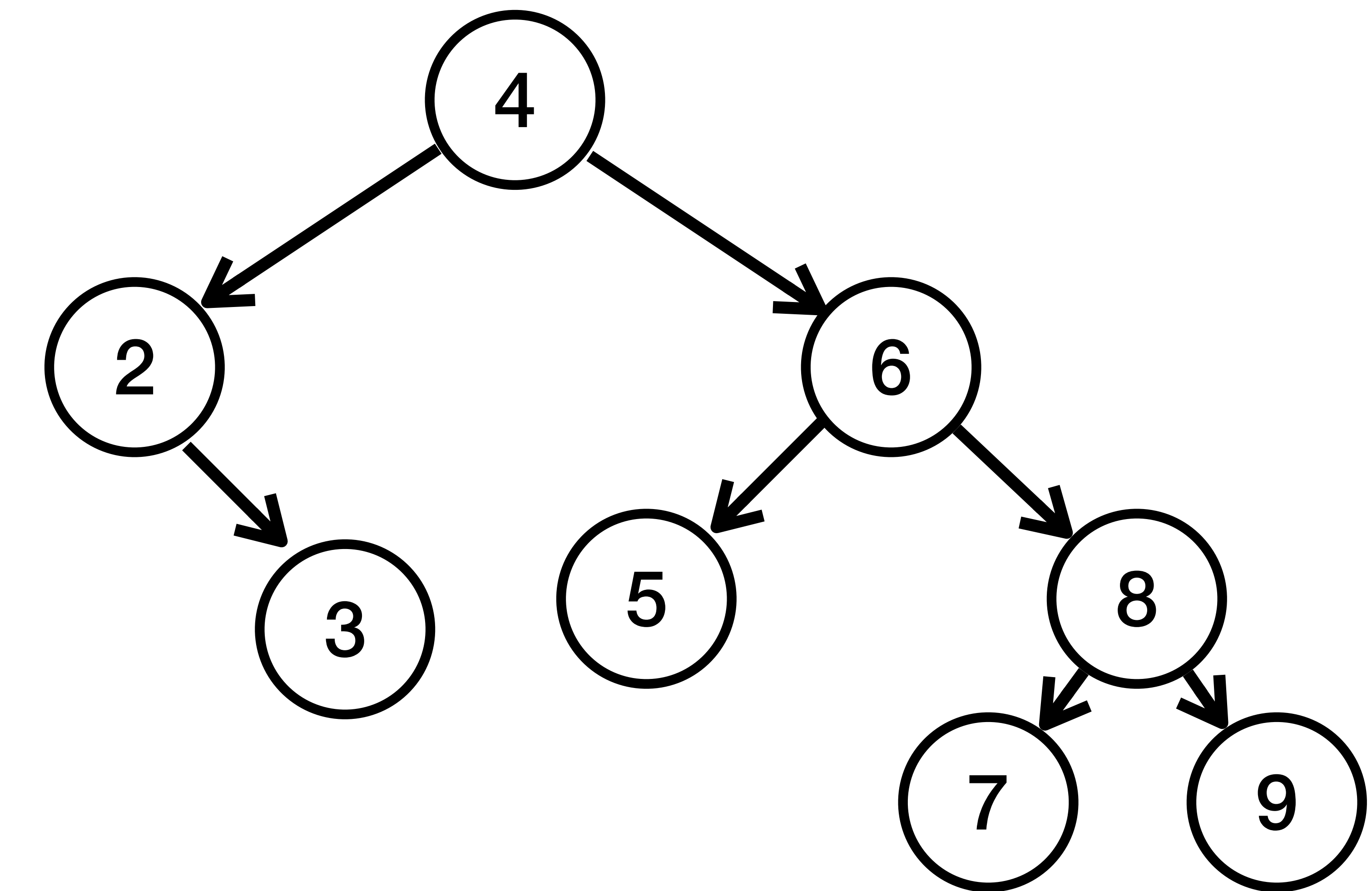
균형잡힌 이진트리

트리 균형 잡기 (Balancing)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음
  - 균형이 깨질 경우 노드 회전(rotation)을 통해 균형을 잡아야 함



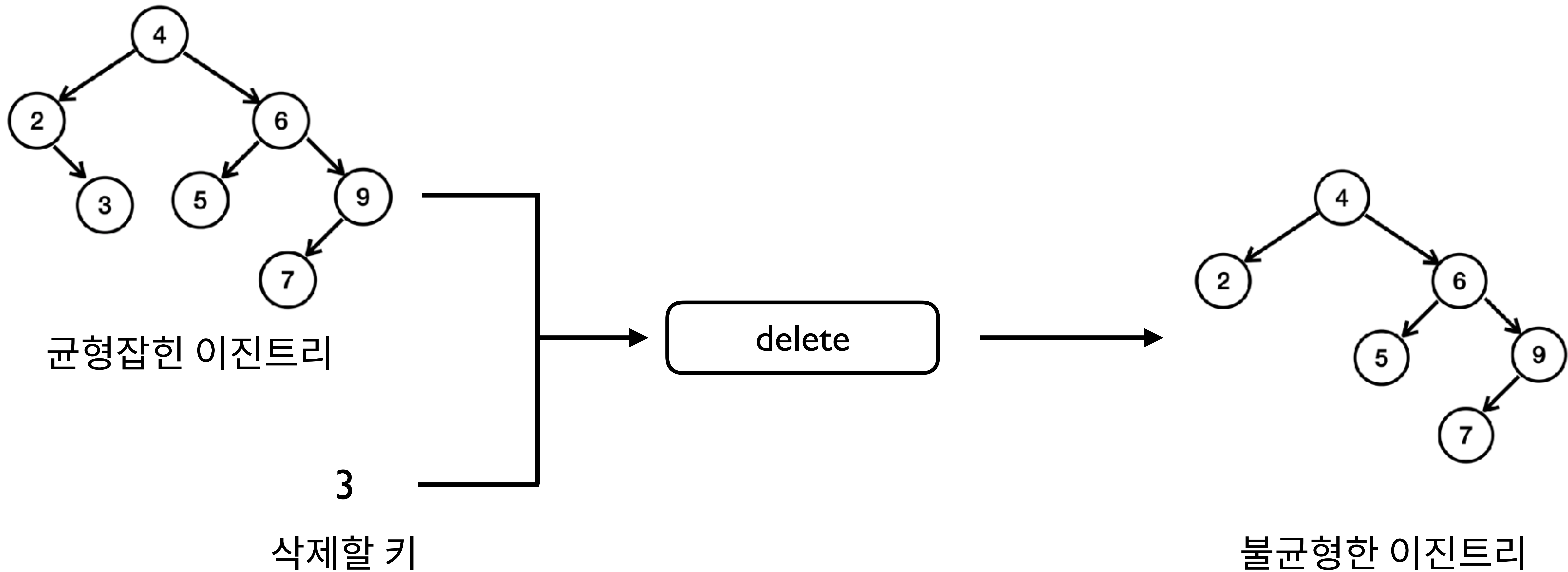
불균형한 이진트리



균형잡힌 이진트리

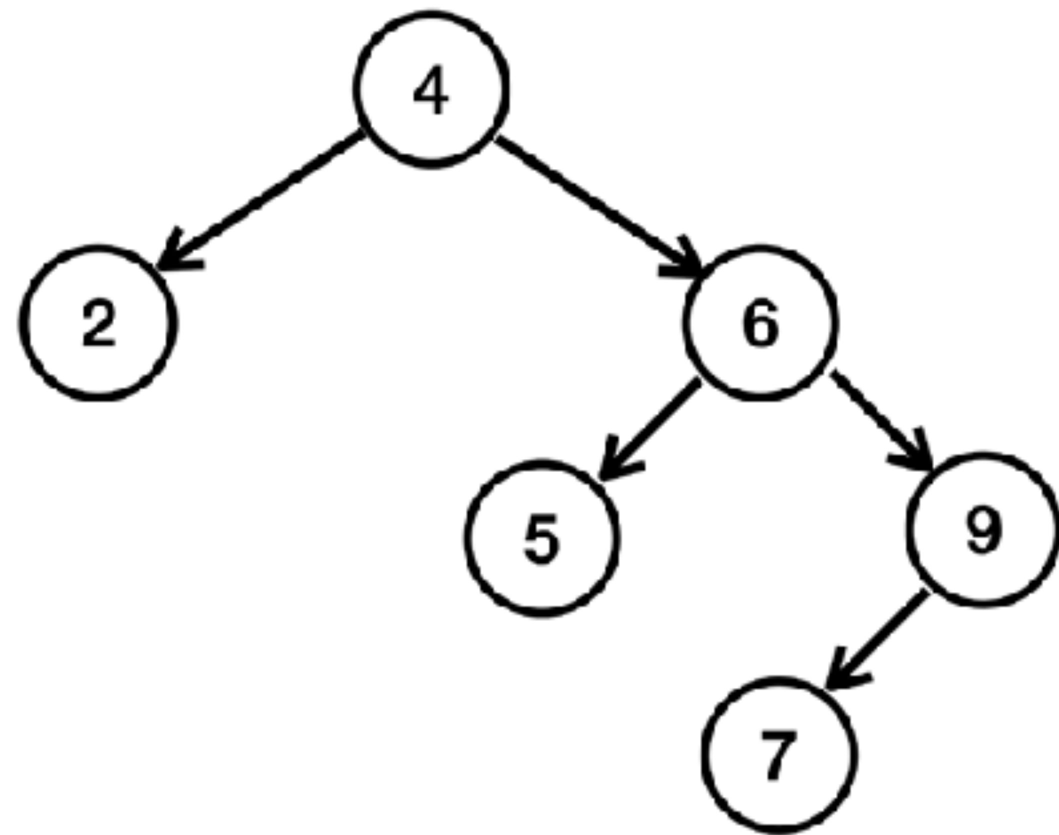
이진 탐색 트리에서의 삭제(delete)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음



트리 균형 잡기 (Balancing Trees)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음
  - 균형이 깨질 경우 노드 회전(rotation)을 통해 균형을 잡아야 함



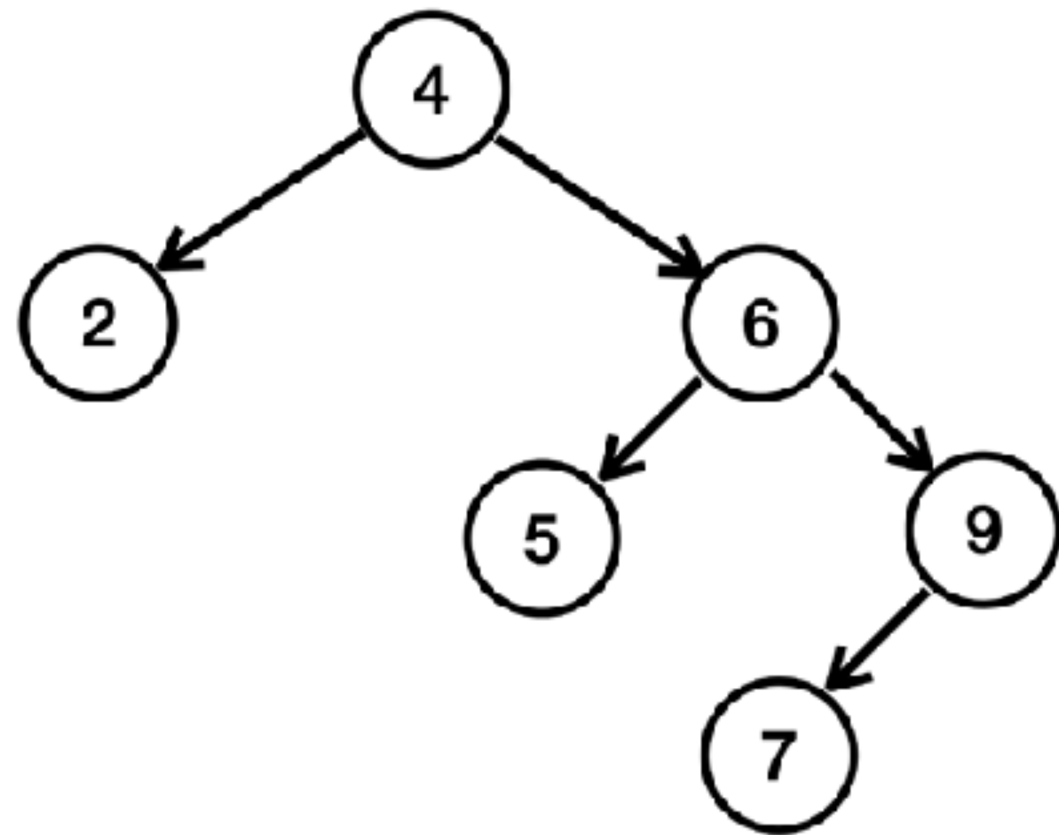
balancing

불균형한 이진트리

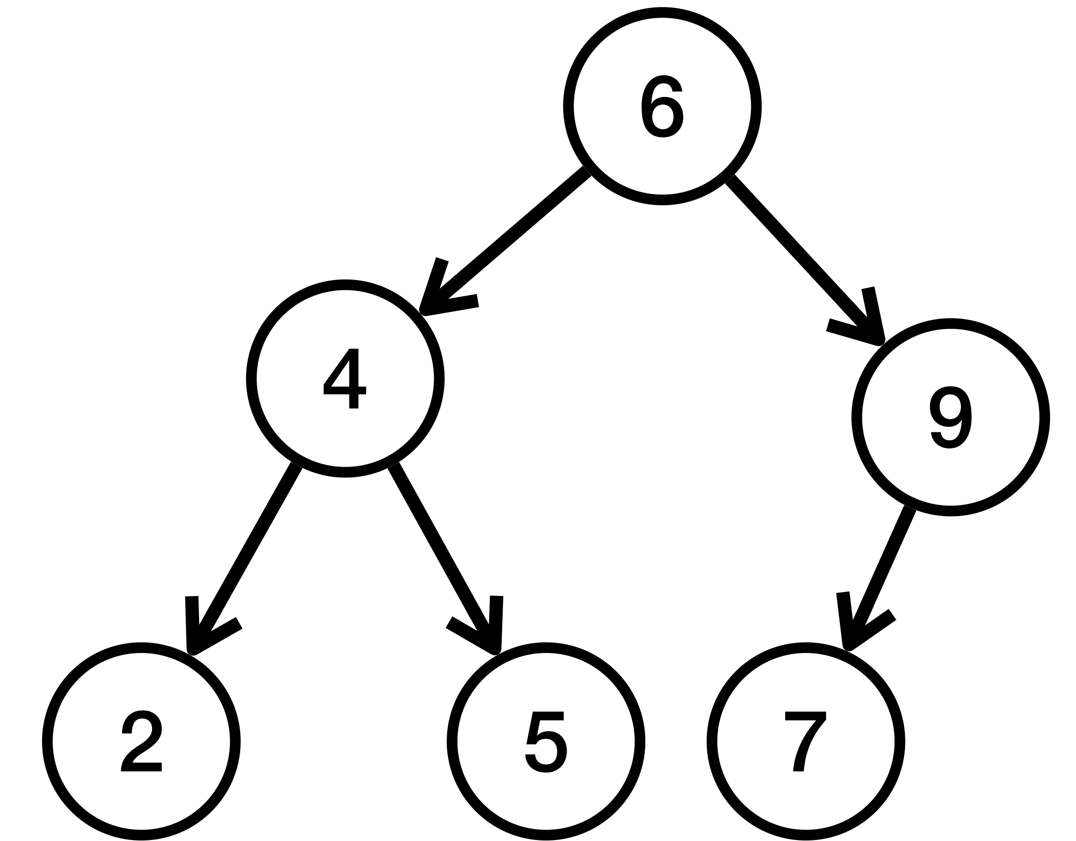
균형잡힌 이진트리

트리 균형 잡기 (Balancing Trees)

- 노드를 삽입 또는 제거할 때 트리의 균형이 깨질 수 있음
  - 균형이 깨질 경우 노드 회전(rotation)을 통해 균형을 잡아야 함



balancing



불균형한 이진트리

불균형한 이진트리

트리 균형 잡기 (Balancing Trees)

- 노드 삽입, 삭제 후 트리 균형잡기가 필요한 네가지 경우
    - LL(Left-Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
    - RR(Right-Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우
    - LR(Left-Right) 케이스: 불균형 노드의 왼쪽(L) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우
    - RL(Right-Left) 케이스: 불균형 노드의 오른쪽(R) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우

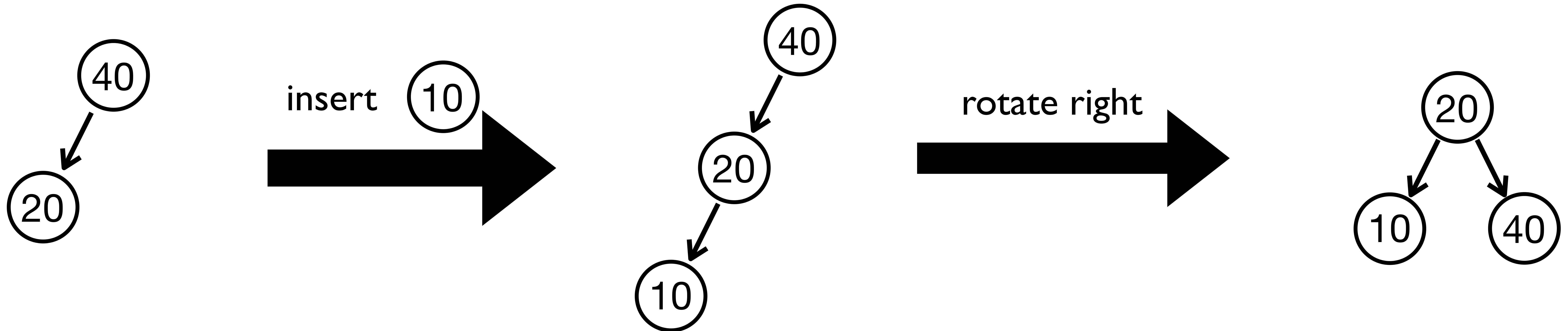
트리 균형잡기가 필요한 네가지 경우

- LL(Left - Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우

40
20
insert 10
40
20
10

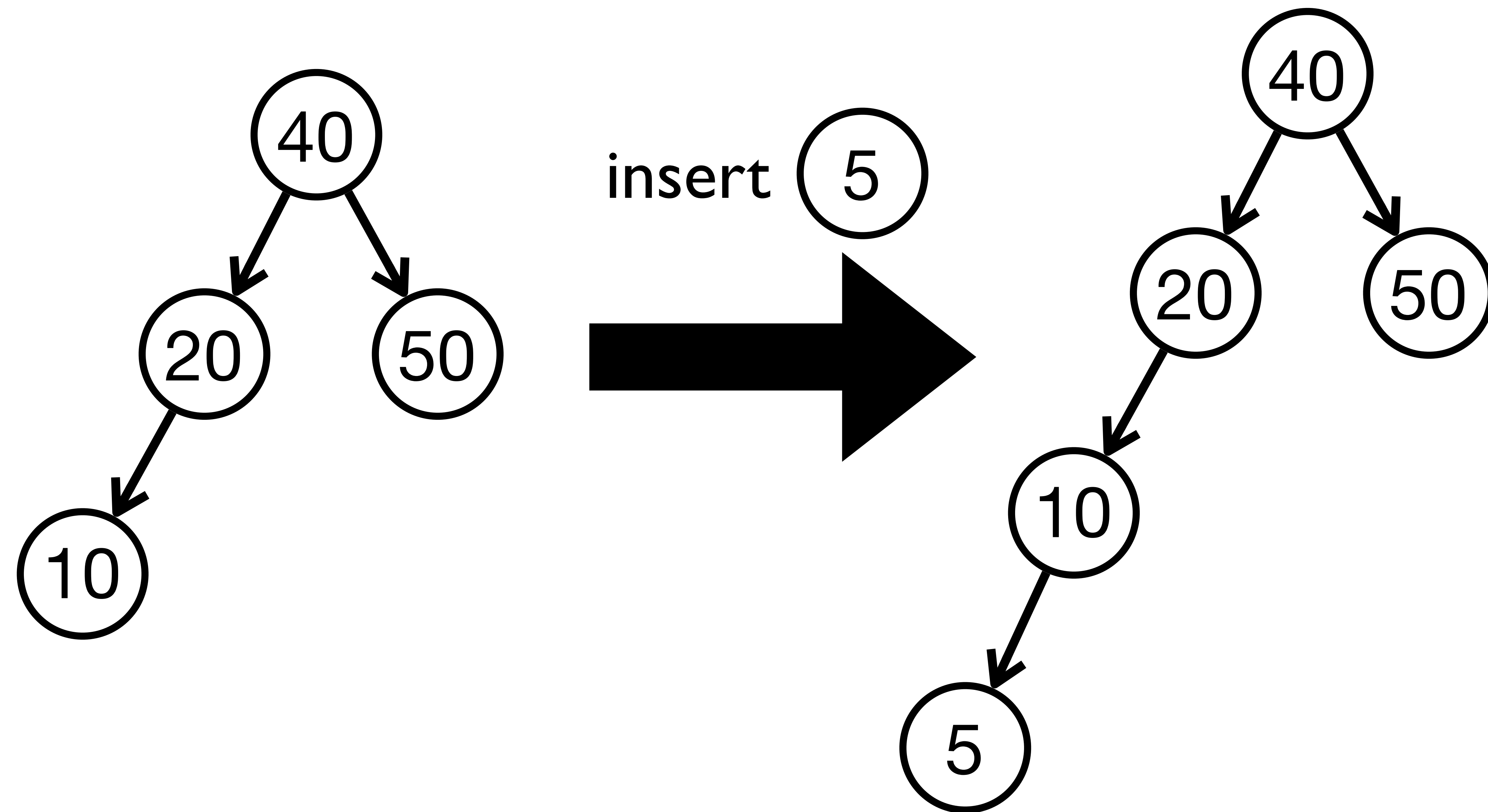
트리 균형잡기가 필요한 네가지 경우

- LL(Left - Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 오른쪽으로 회전



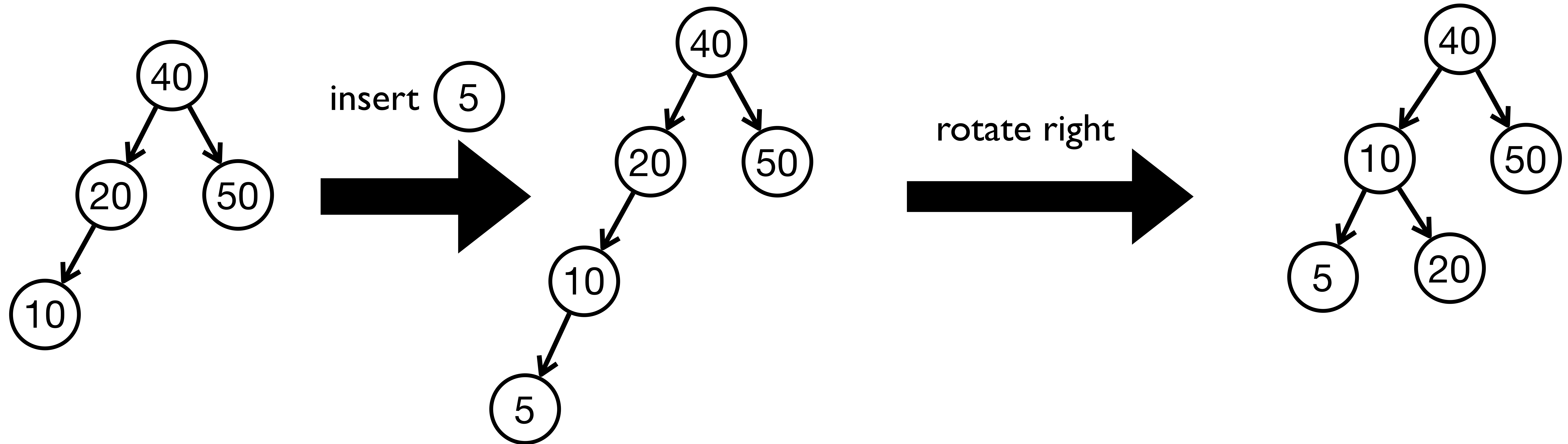
트리 균형잡기가 필요한 네가지 경우

- LL(Left - Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 오른쪽으로 회전



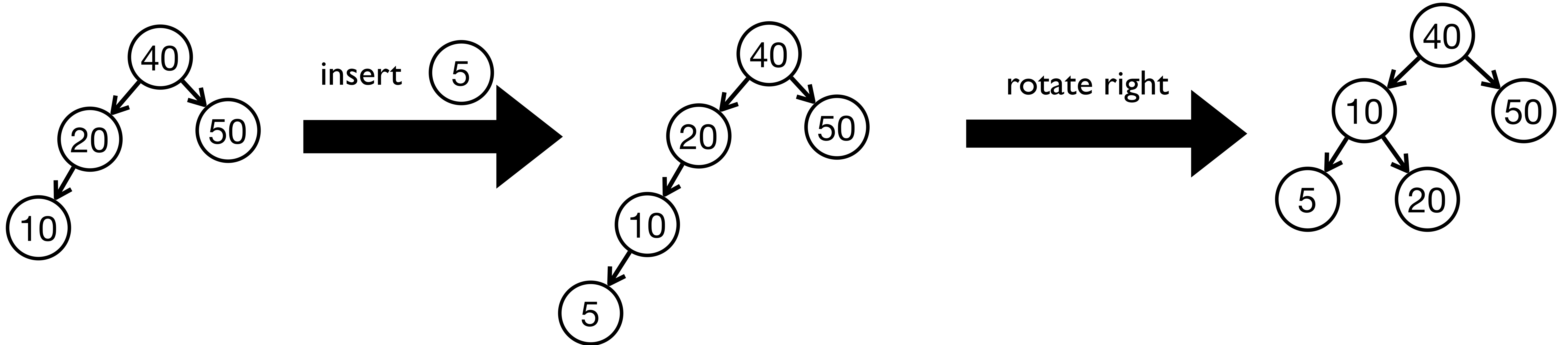
트리 균형잡기가 필요한 네가지 경우

- LL(Left - Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 오른쪽으로 회전



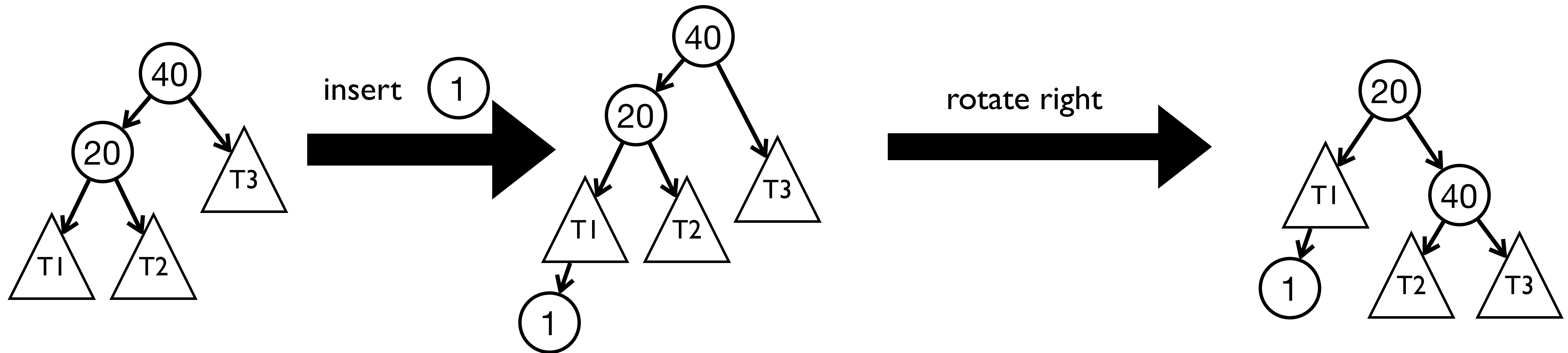
트리 균형잡기가 필요한 네가지 경우

- LL(Left - Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 오른쪽으로 회전



트리 균형잡기가 필요한 네가지 경우

- LL(Left - Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
    - 불균형이 발생한 노드를 오른쪽으로 회전

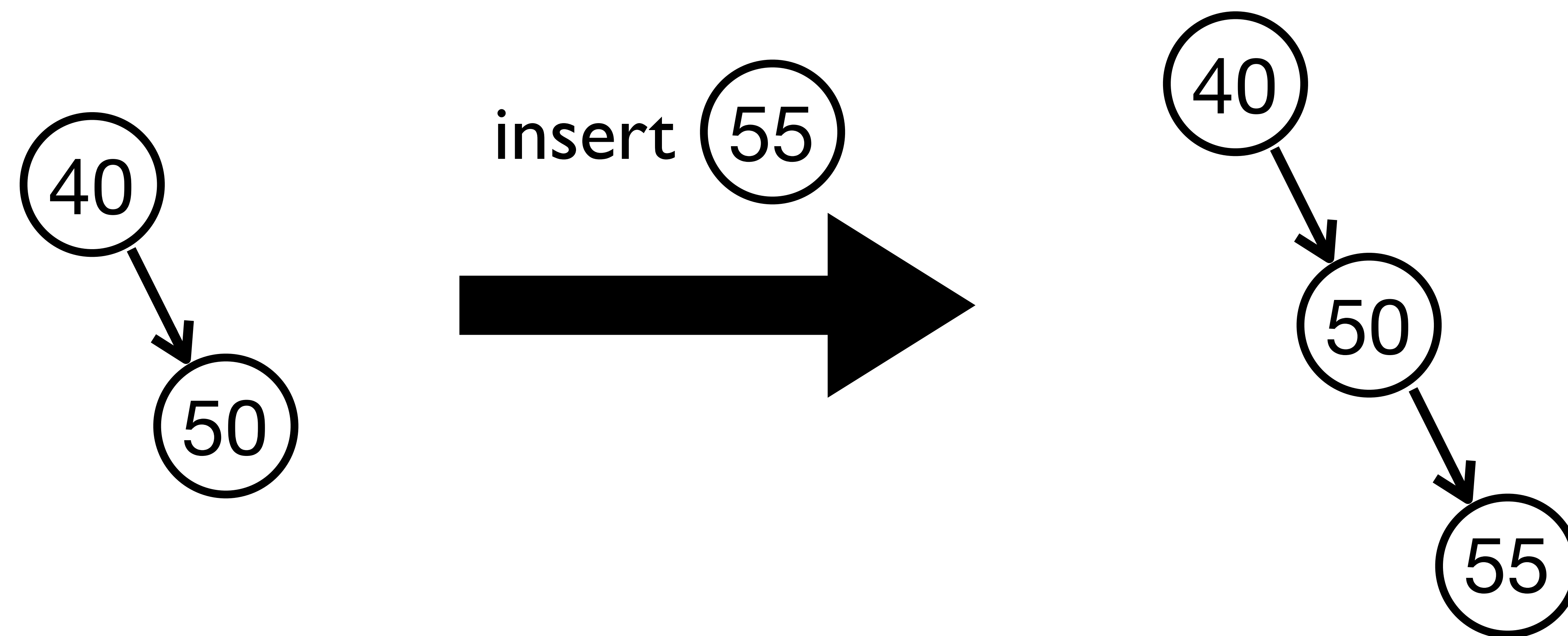


트리 균형잡기가 필요한 네가지 경우

- LL(Left - Left) 케이스: 불균형 노드의 왼쪽(L) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 오른쪽으로 회전

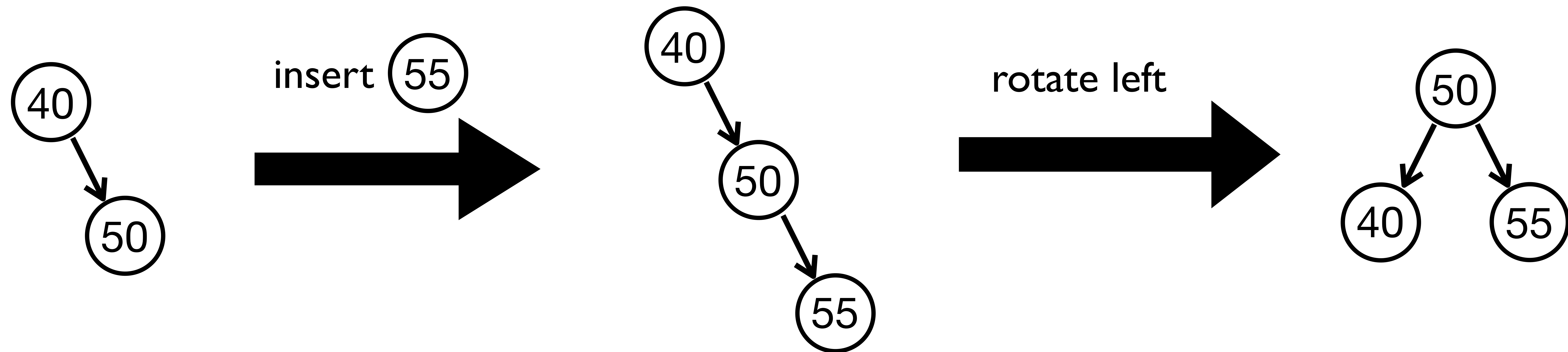
트리 균형잡기가 필요한 네가지 경우

- RR(Right - Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른(R) 서브트리가 너무 깊어 불균형이 발생한 경우



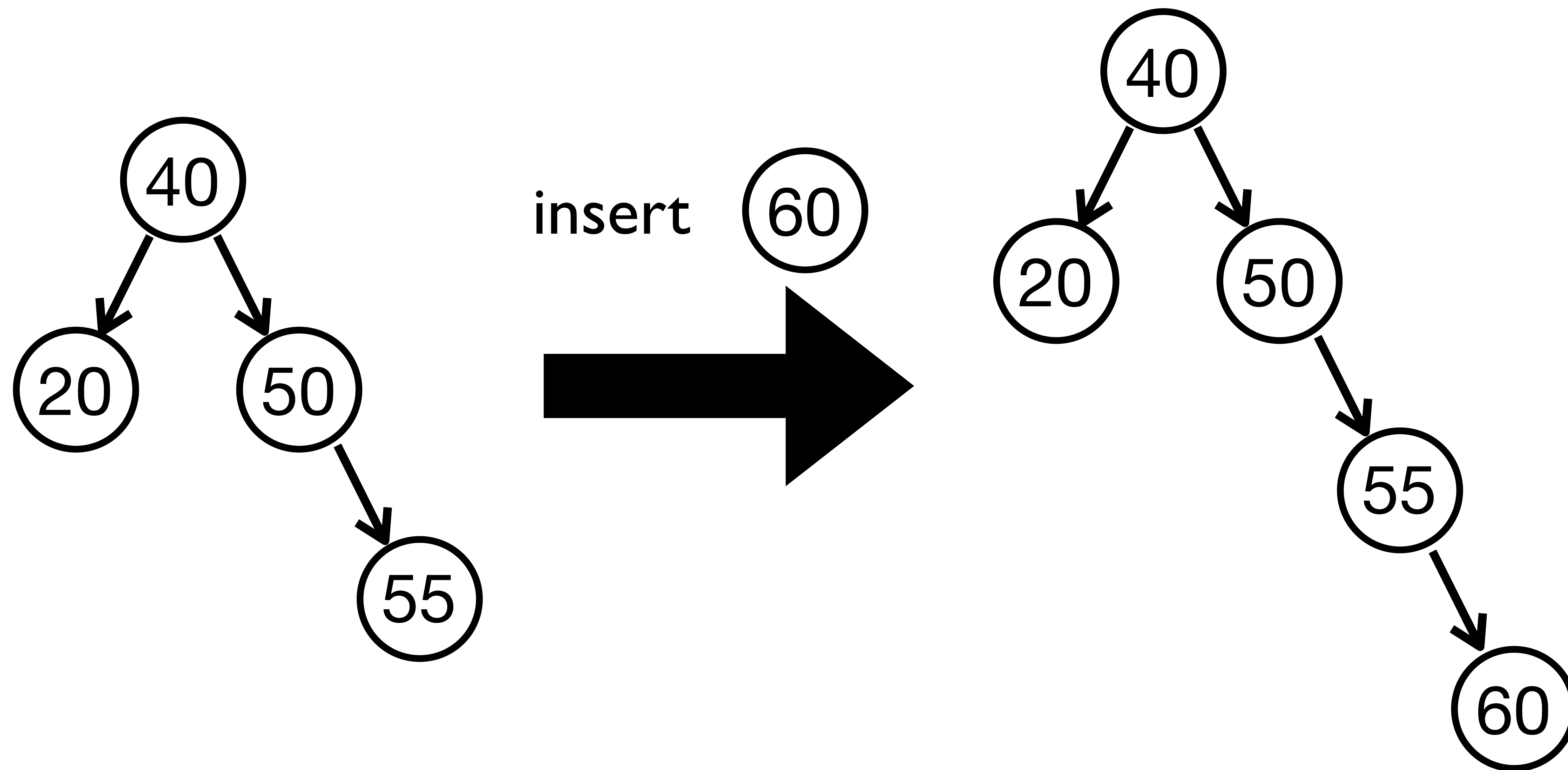
트리 균형잡기가 필요한 네가지 경우

- RR(Right - Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른(R) 서브트리가 너무 깊어 불균형이 발생한 경우
    - 불균형이 발생한 노드를 왼쪽으로 회전



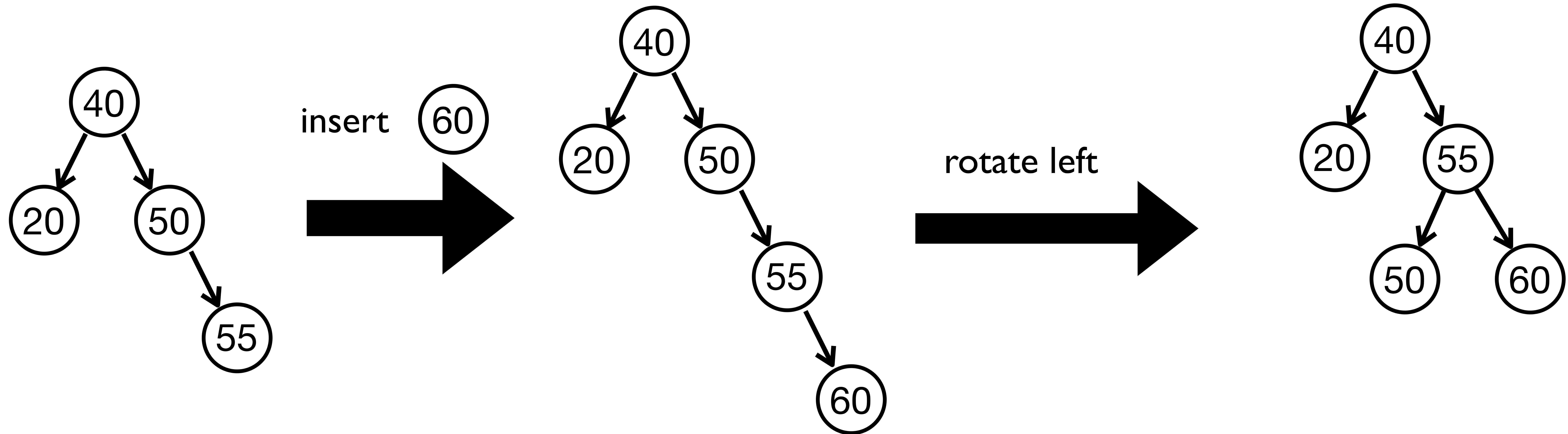
트리 균형잡기가 필요한 네가지 경우

- RR(Right - Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 왼쪽으로 회전



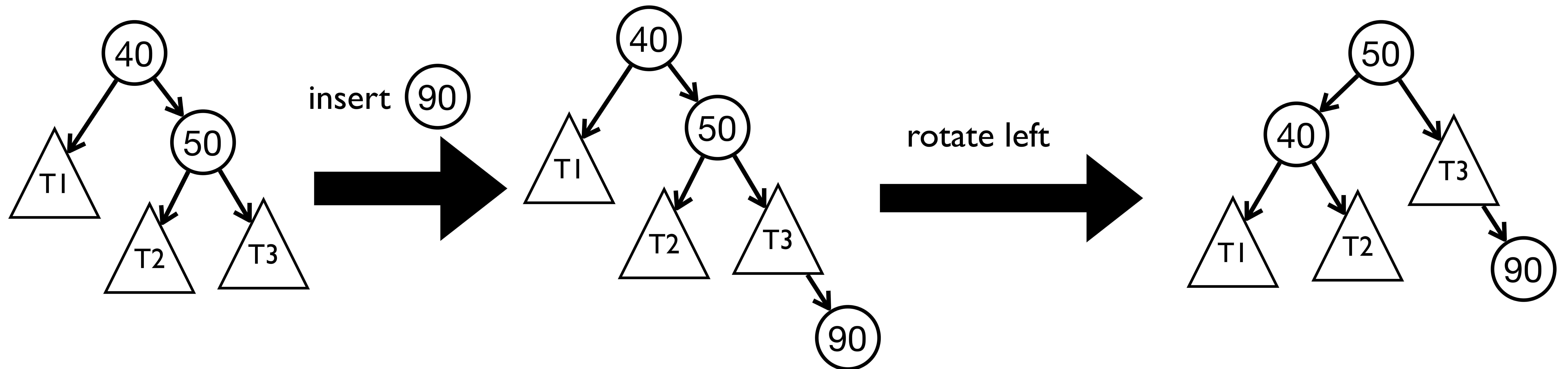
트리 균형잡기가 필요한 네가지 경우

- RR(Right - Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 왼쪽으로 회전



트리 균형잡기가 필요한 네가지 경우

- RR(Right - Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른(R) 서브트리가 너무 깊어 불균형이 발생한 경우
    - 불균형이 발생한 노드를 왼쪽으로 회전

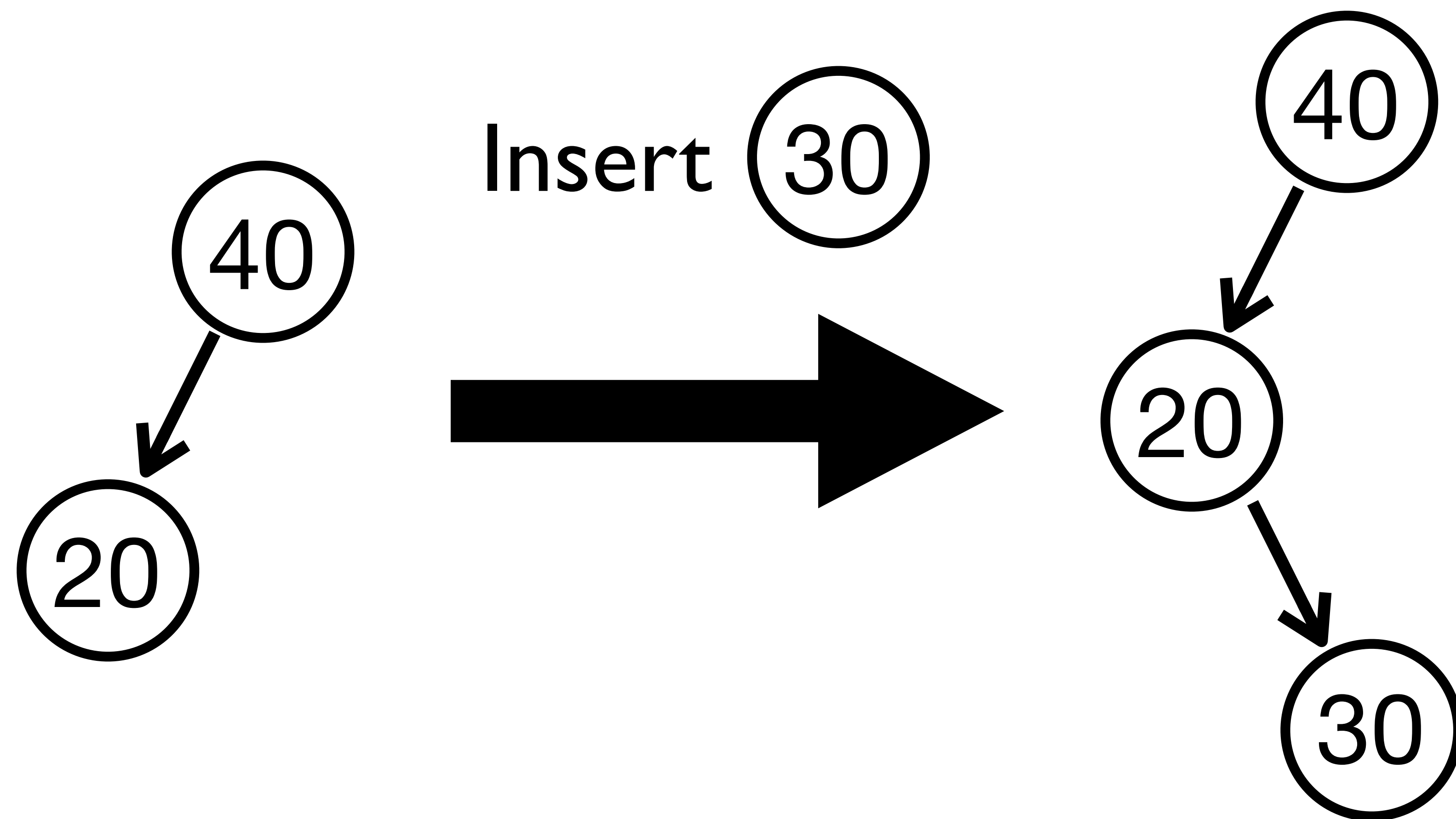


트리 균형잡기가 필요한 네가지 경우

- RR(Right - Right) 케이스: 불균형 노드의 오른쪽(R) 자식의 오른(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드를 왼쪽으로 회전

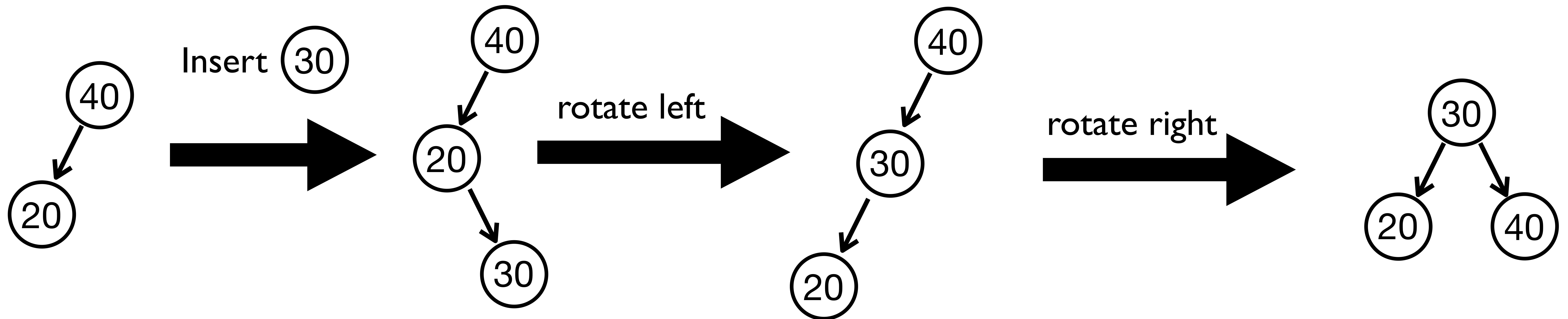
트리 균형잡기가 필요한 네가지 경우

- LR(Left - Right) 케이스: 불균형 노드의 왼쪽(L) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우



트리 균형잡기가 필요한 네가지 경우

- LR(Left - Right) 케이스: 불균형 노드의 왼쪽(L) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우
    - 불균형이 발생한 노드의 왼쪽 자식노드를 왼쪽으로 회전 후 분균형이 발생한 노드를 오른쪽으로 회전



트리 균형잡기가 필요한 네가지 경우

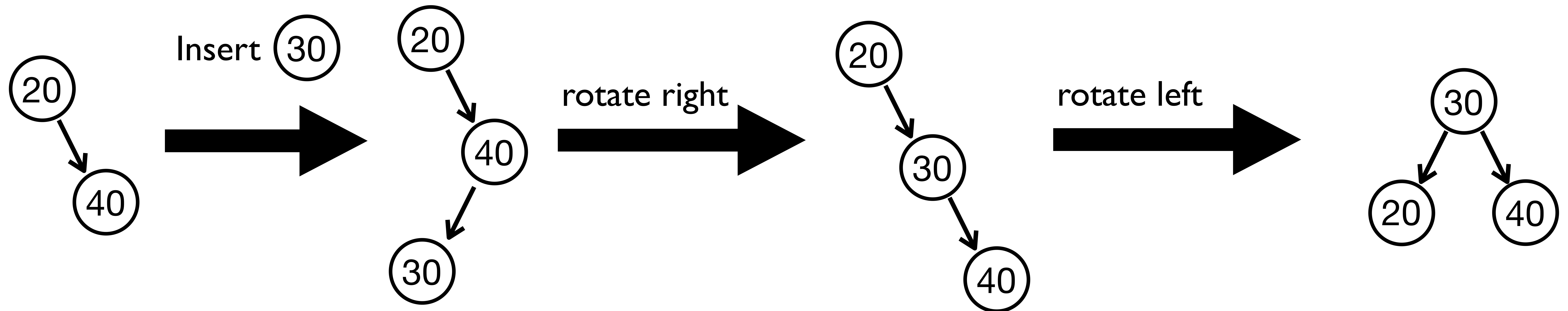
- LR(Left - Right) 케이스: 불균형 노드의 왼쪽(L) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드의 왼쪽 자식노드를 왼쪽으로 회전 후 분균형이 발생한 노드를 오른쪽으로 회전

트리 균형잡기가 필요한 네가지 경우

- LR(Left - Right) 케이스: 불균형 노드의 왼쪽(L) 자식의 오른쪽(R) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드의 왼쪽 자식노드를 왼쪽으로 회전 후 분균형이 발생한 노드를 오른쪽으로 회전

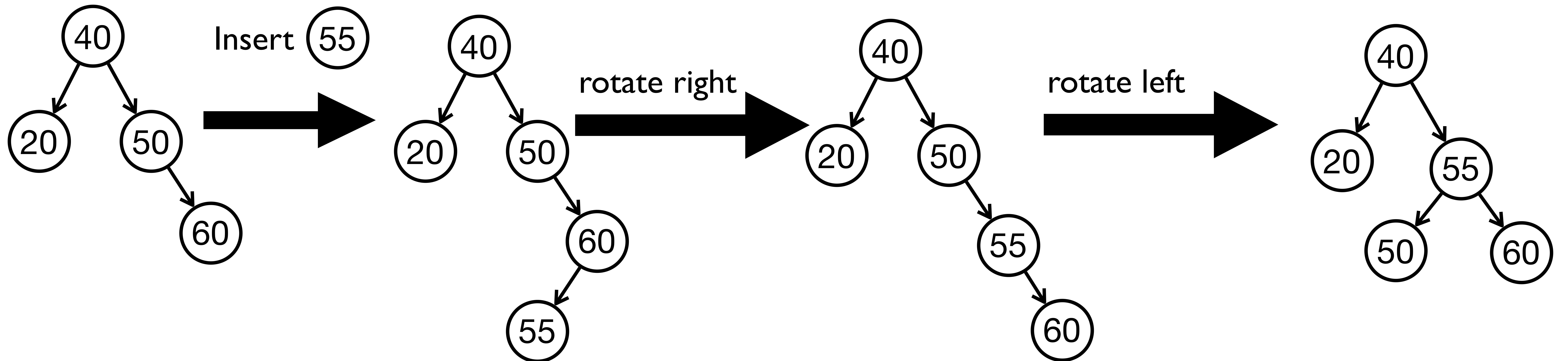
트리 균형잡기가 필요한 네가지 경우

- RL(Right - Left) 케이스: 불균형 노드의 오른쪽(R) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드의 오른쪽 자식노드를 오른쪽으로 회전 후 분균형이 발생한 노드를 왼쪽으로 회전



트리 균형잡기가 필요한 네가지 경우

- RL(Right - Left) 케이스: 불균형 노드의 오른쪽(R) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
    - 불균형이 발생한 노드의 오른쪽 자식노드를 오른쪽으로 회전 후 분균형이 발생한 노드를 왼쪽으로 회전



트리 균형잡기가 필요한 네가지 경우

- RL(Right - Left) 케이스: 불균형 노드의 오른쪽(R) 자식의 왼쪽(L) 서브트리가 너무 깊어 불균형이 발생한 경우
  - 불균형이 발생한 노드의 오른쪽 자식노드를 오른쪽으로 회전 후 분균형이 발생한 노드를 왼쪽으로 회전

Examples

- 빈 AVL 트리에 다음과 같은 key값들로 노드들을 삽입(insert)하였을 때 결과 AVL트리를 그리시오

(1) 10 -> 20 -> 30 -> 40 -> 50

(2) 30 -> 50 -> 70 -> 20 -> 10

(3) 70 -> 30 -> 50 -> 20 -> 40 -> 35

(4) 30 -> 40 -> 50 -> 45 -> 70 -> 60

AVL 트리

- AVL 트리의 추상 자료형:
  - insert(bst, key): AVL 트리의 특성을 유지하면서 key를 키로 가지는 새로운 노드를 삽입함
    - 트리의 균형이 무너질 경우 불균형한 노드를 회전시켜 균형을 잡음
  - deleteNode(bst, key): AVL 트리의특성을 유지하면서 key를 키로 가지는 노드를 삭제함
    - 트리의 균형이 무너질 경우 불균형한 노드를 회전시켜 균형을 잡음
  - findMin(bst, n): AVL 트리에서 키값이 가장 작은 노드를 찾아서 반환함
  - search(bst, key): AVL 트리에서 키값이 key인 노드를 찾아 반환함
  - traversal(bst): AVL 트리를 구성하는 노드들의 키값들을 출력함
  - height(bst): AVL 트리의 높이를 반환함

insert

- insert: AVL 트리의 특성을 유지하면서 새로운 노드를 AVL 트리에 삽입 후 루트 노드를 반환

```
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 balancing(root)
end procedure
```

deleteNode

- deleteNode: AVL 트리의특성을 유지하면서 노드 key를 키값으로 가지는 노드를 AVL 트리에서 삭제후 루트 노드를 반환

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

balancing

- balancing: AVL tree의 균형을 맞춘 후 루트 노드를 반환

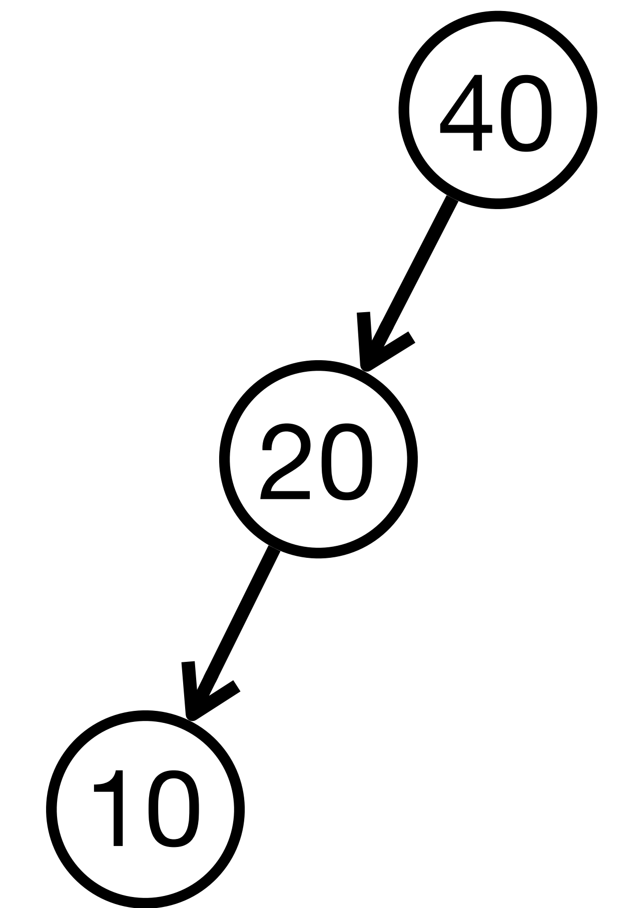
```
procedure balancing(root)
  if (getBalance(root) > 1) and (getBalance(root->left) ≥ 0) then
    return rotateRight(root)
  elif (getBalance(root) > 1) and (getBalance(root->left) < 0) then
    root.left ← rotateLeft(root.left)
    return rotateRight(root)
  elif (getBalance(root) < -1) and (getBalance(root->left) ≤ 0) then
    return rotateLeft(root)
  elif (getBalance(root) < -1) and (getBalance(root->left) > 0) then
    root.right ← rotateRight(root.right)
    return rotateLeft(root)
  return root
```

balancing

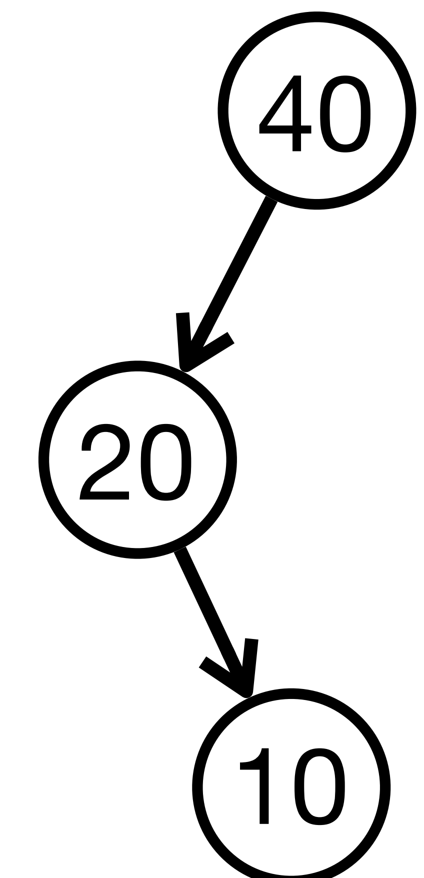
- balancing: AVL tree의 균형을 맞춘 후 루트 노드를 반환

```
procedure balancing(root)
  if (getBalance(root) > 1) and (getBalance(root->left) ≥ 0) then
    return rotateRight(root)
  elif (getBalance(root) > 1) and (getBalance(root->left) < 0) then
    root.left ← rotateLeft(root.left)
    return rotateRight(root)
  elif (getBalance(root) < -1) and (getBalance(root->left) ≤ 0) then
    return rotateLeft(root)
  elif (getBalance(root) < -1) and (getBalance(root->left) > 0) then
    root.right ← rotateRight(root.right)
    return rotateLeft(root)
  return root
```

LL(Left-Left) 케이스



LR(Left-Right) 케이스

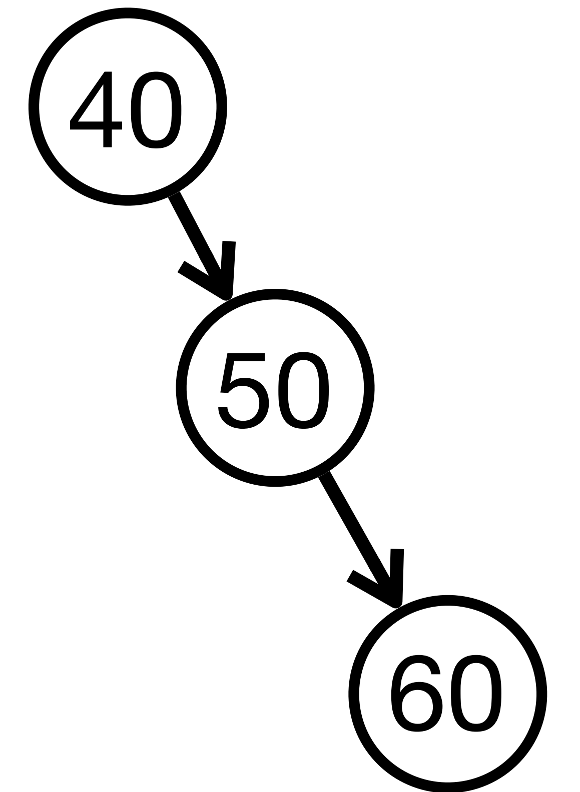


balancing

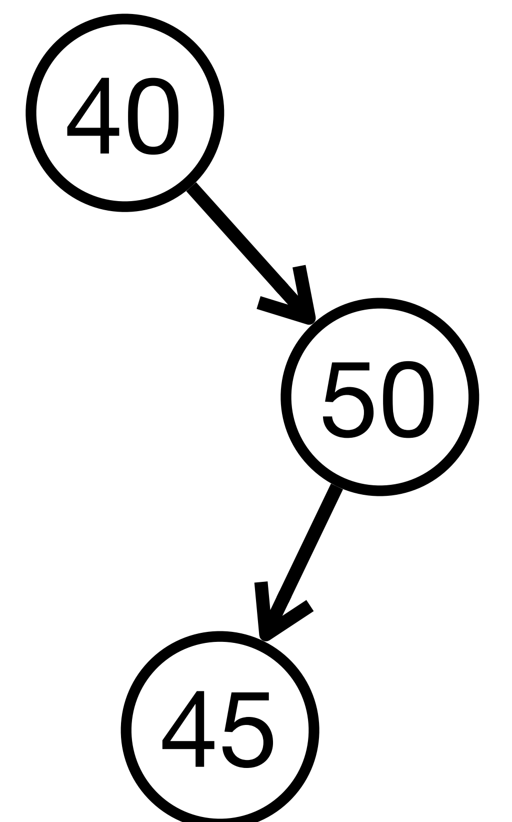
- balancing: AVL tree의 균형을 맞춘 후 루트 노드를 반환

```
procedure balancing(root)
  if (getBalance(root) > 1) and (getBalance(root->left) ≥ 0) then
    return rotateRight(root)
  elif (getBalance(root) > 1) and (getBalance(root->left) < 0) then
    root.left ← rotateLeft(root.left)
    return rotateRight(root)
  elif (getBalance(root) < -1) and (getBalance(root->left) ≤ 0) then
    return rotateLeft(root)
  elif (getBalance(root) < -1) and (getBalance(root->left) > 0) then
    root.right ← rotateRight(root.right)
    return rotateLeft(root)
  return root
```

RR(Right-Right) 케이스



RL(Right-Left) 케이스



AVL 탐색 트리의 균형잡기

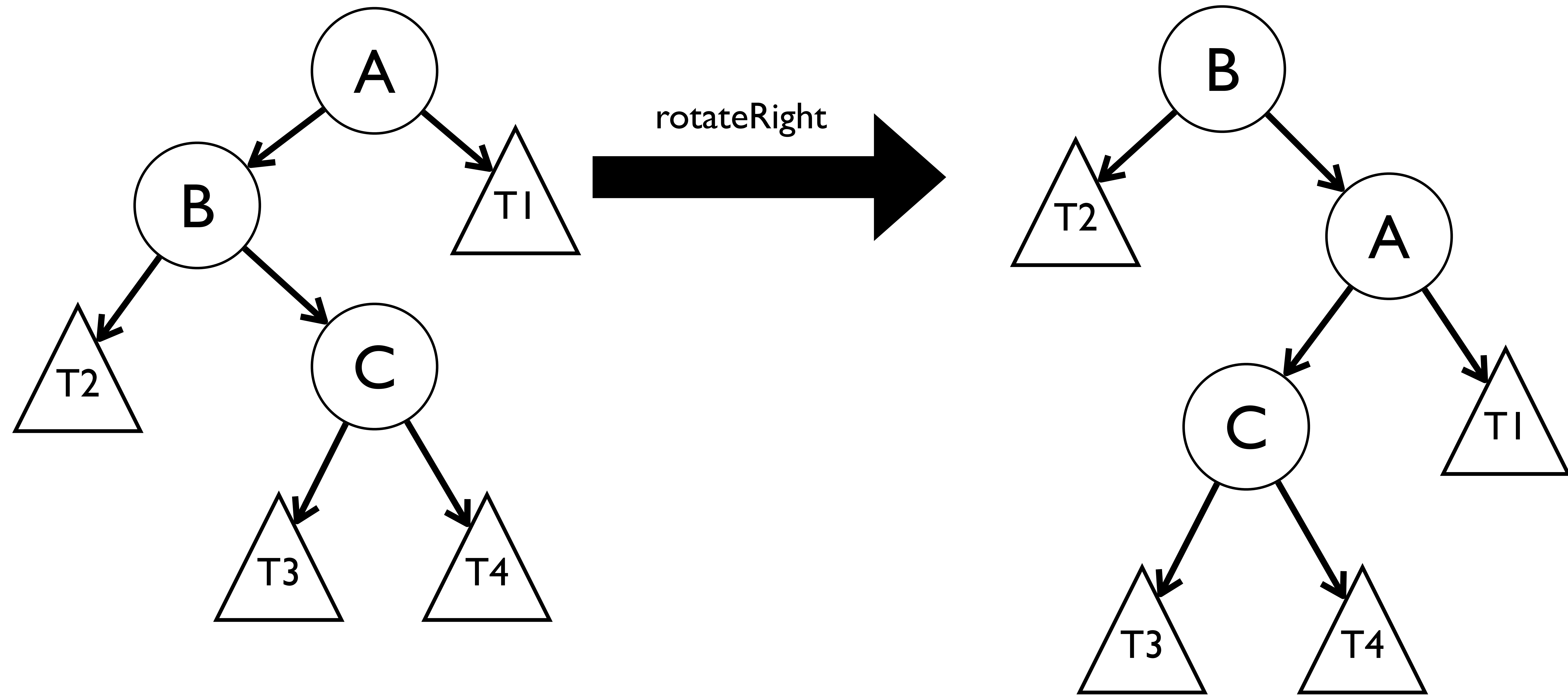
- getBalance: AVL tree의 균형인수를 반환

```
procedure getBalance(root)
  if root = NULL then
    return 0
  end if
  return height(root.left) - height(root.right)
```

AVL 탐색 트리의 균형잡기

- rotateRight: AVL tree의 루트노드를 기준으로 오른쪽으로 회전 후 AVL 트리의 루트노드를 반환

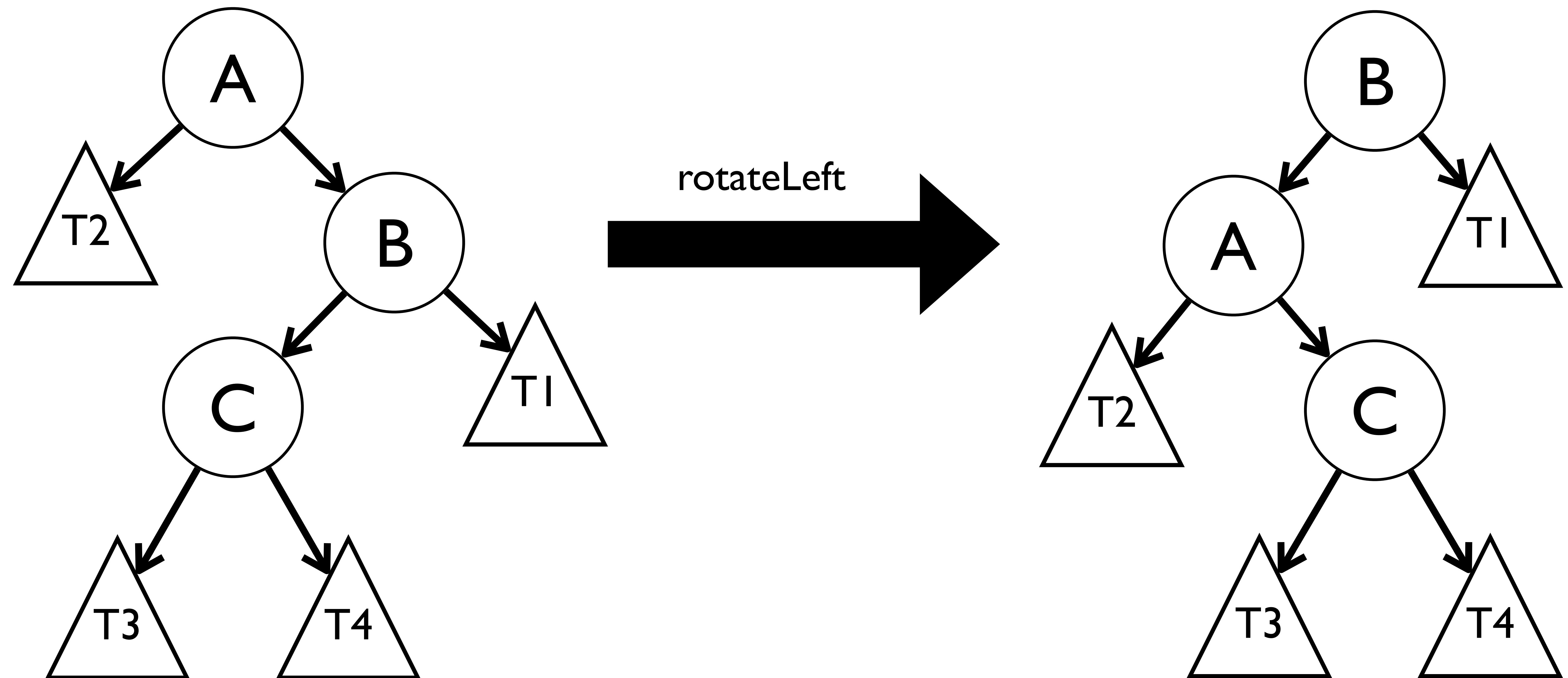
```
procedure rotateRight(root)
  node1 ← root.left
  node2 ← node1.right
  node1.right ← root
  root.left ← node2
  return node1
end procedure
```



AVL 탐색 트리의 균형잡기

- rotateLeft: AVL tree의 루트노드를 기준으로 왼쪽으로 회전 후 AVL 트리의 루트노드를 반환

```
procedure rotateLeft(root)
  node1 ← root.right
  node2 ← node1.left
  node1.left ← root
  root.right ← node2
  return node1
end procedure
```



AVL 탐색 트리의 구현

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

- `Node* create()`:
  - 비어있는 AVL 트리를 생성 후 반환함
- `Node* insert(Node* root, int key)`:
  - AVL 트리의 특성을 유지하면서 key를 키값으로 가지는 노드를 AVL 트리에 삽입 후 루트 노드를 반환
- `Node* findMin(Node* node)`:
  - AVL 트리에서 key값이 최소인 노드를 반환함
- `Node* deleteNode(Node* root, int key)`:
  - AVL 트리의 특성을 유지하면서 노드 key를 키값으로 가지는 노드를 AVL 트리에서 삭제 후 루트 노드를 반환
- `Node* search(Node* root, int key)`:
  - AVL 트리에서 키값이 key인 노드를 찾아 반환함
- `void traversal(Node* root)`:
  - AVL 트리를 구성하는 노드들의 키값들을 오름차순으로 출력함
- `int height(Node* root)`:
  - AVL 트리의 높이를 반환함

AVL 트리의 구현

- AVL 트리의 노드는 다음과 같은 정보를 가짐

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

- create: 비어있는 AVL 트리를 생성 후 반환함

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

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

AVL 트리의 구현

- search: AVL 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) {

}
```

AVL 트리의 구현

- findMin: AVL 트리에서 key값이 최소인 노드를 반환함

```
procedure findMin(root)
    current ← root
    while (current ≠ NULL) and (current.left ≠ NULL)  do
        current ← current.left
    end while
    return current
```

```
Node* findMin(Node* node) {

}
```

AVL 트리의 구현

- height: AVL 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
```

```
int height(Node* root) {

}
```

AVL 트리의 구현

- traversal: AVL 트리를 구성하는 노드들의 키값들을 오름차순으로 출력함

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

```
void traversal(Node* root) {

}
```

AVL 트리의 구현

- insert: AVL 트리의 특성을 유지하면서 새로운 노드를 AVL 트리에 삽입 후 루트 노드를 반환

```
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 balancing(root)
end procedure
```

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

}
```

AVL 트리의 구현

- deleteNode: AVL 트리의특성을 유지하면서 노드 key를 키값으로 가지는 노드를 AVL 트리에서 삭제후 루트 노드를 반환

```
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 ← delete(root.right, node.key)
 end if
 return balancing(root)
end procedure
```

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

}
```

AVL 트리의 구현

- balancing: AVL tree의 균형을 맞춘 후 루트 노드를 반환

```
procedure balancing(root)
  if (getBalance(root) > 1) and (getBalance(root->left) ≥ 0) then
    return rotateRight(root)
  elif (getBalance(root) > 1) and (getBalance(root->left) < 0) then
    root.left ← rotateLeft(root.left)
    return rotateRight(root)
  elif (getBalance(root) < -1) and (getBalance(root->left) ≤ 0) then
    return rotateLeft(root)
  elif (getBalance(root) < -1) and (getBalance(root->left) > 0) then
    root.right ← rotateRight(root.right)
    return rotateLeft(root)
  return root
```

```
Node* balancing(Node* root) {

}
```

AVL 트리의 구현

- getBalancing: AVL 트리의 균형인수를 반환

```
procedure getBalance(root)
  if root = NULL then
    return 0
  end if
  return height(root.left) - height(root.right)
```

```
int getBalance(Node* root) {

}
```

AVL 트리의 구현

- rotateRight: AVL tree의 루트노드를 기준으로 오른쪽으로 회전 후 AVL 트리의 루트노드를 반환

```
procedure rotateRight(root)
  node1 ← root.left
  node2 ← node1.right
  node1.right ← root
  root.left ← node2
  return node1
end procedure
```

```
Node* rotateRight(Node* root) {

}
```

AVL 트리의 구현

- rotateLeft: AVL tree의 루트노드를 기준으로 왼쪽으로 회전 후 AVL 트리의 루트노드를 반환

```
procedure rotateLeft(root)
  node1 ← root.right
  node2 ← node1.left
  node1.left ← root
  root.right ← node2
  return node1
end procedure
```

```
Node* rotateLeft(Node* root) {

}
```

Example

```c
#include <stdio.h>
#include "AVL.h"

int main() {
    Node *root = create();
    for (int i = 0; i < 8; i++) {
        root = insert(root, i*10);
        printf("Height : %d\n", height(root));
    }

    Node *node = search(root, 70);
    if (node == NULL) {
        printf("Node not found\n");
    } else {
        printf("Node: %d\n", node->key);
    }

    printf("Traversal: ");
    traversal(root);
    printf("\n");
    for (int i = 0; i < 8; i++) {
        root = deleteNode(root, i*10);
        printf("Height : %d\n", height(root));
    }
    return 0;
}
```

마무리

- AVL 탐색 트리: 항상 균형을 이루는 이진 탐색 트리
- AVL 탐색 트리가 만족해야 할 성질
  - 균형인수 B $(H_L - H_R)$가 -1, 0, 또는 1
    - $B = 1$: 왼쪽 서브트리의 높이가 1 더 큼
    - $B = 0$: 두 서브트리의 높이가 같음
    - $B = -1$: 오른쪽 서브트리의 높이가 1 더 큼
- AVL 탐색 트리의 서브트리도 AVL트리임