Data Structure #8

AVL tree

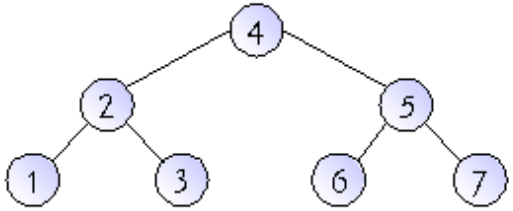
2020년 1학기

Intro.

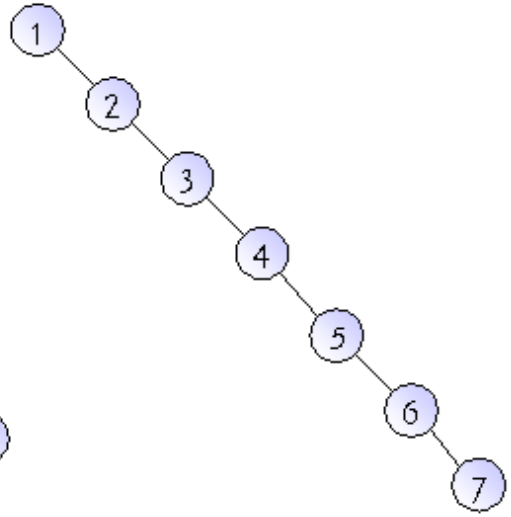
- **실습 주제**
  - AVL: insert
    - LL rotation
    - RR rotation
    - LR rotation
    - RL rotation

AVL 트리

- **AVL 트리**
  - 서브트리의 높이를 적절하게 제어하여 전체 트리가 한쪽으로 늘어지지 않도록 한 이진탐색트리(Binary Search Tree)
  - 균형된 트리를 구성하여 트리의 높이를 줄임 -> 탐색시간을 줄임



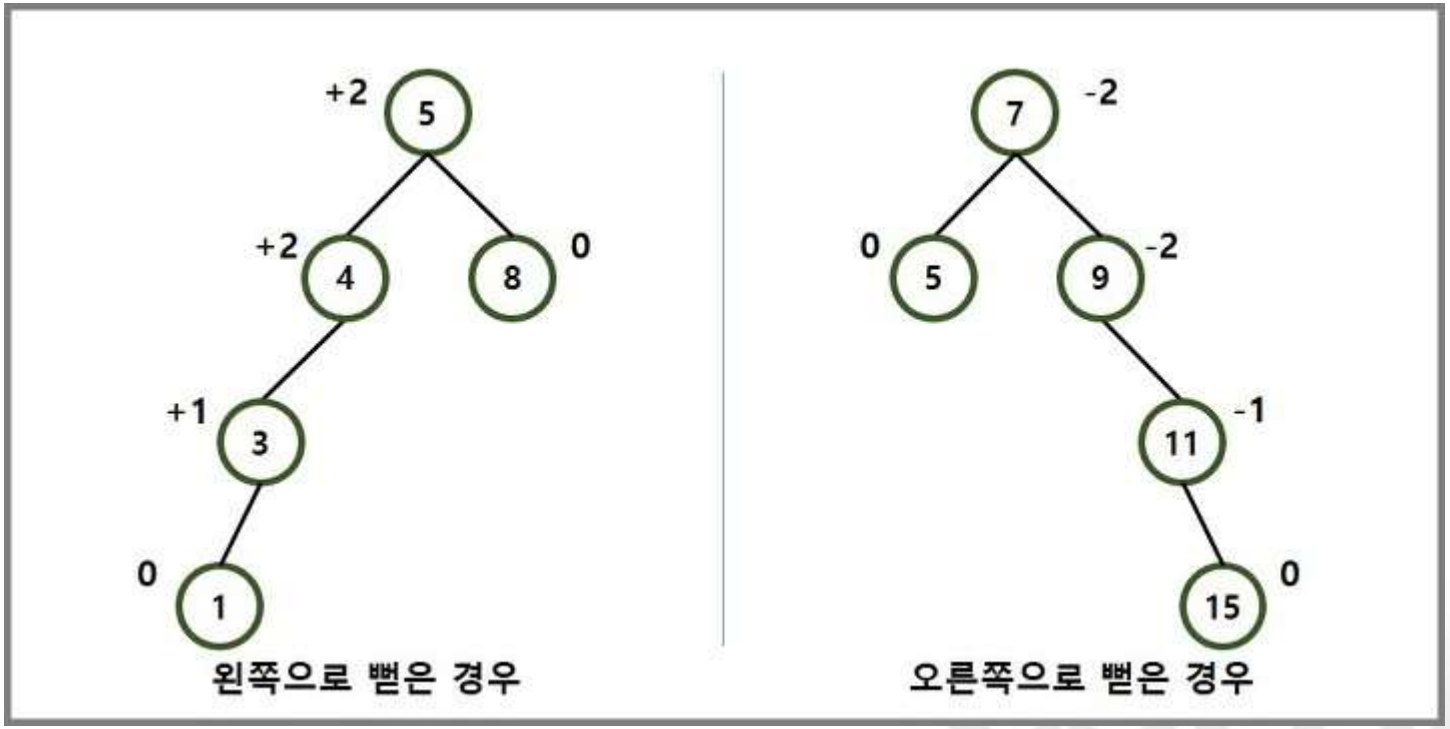
균형트리



불균형트리

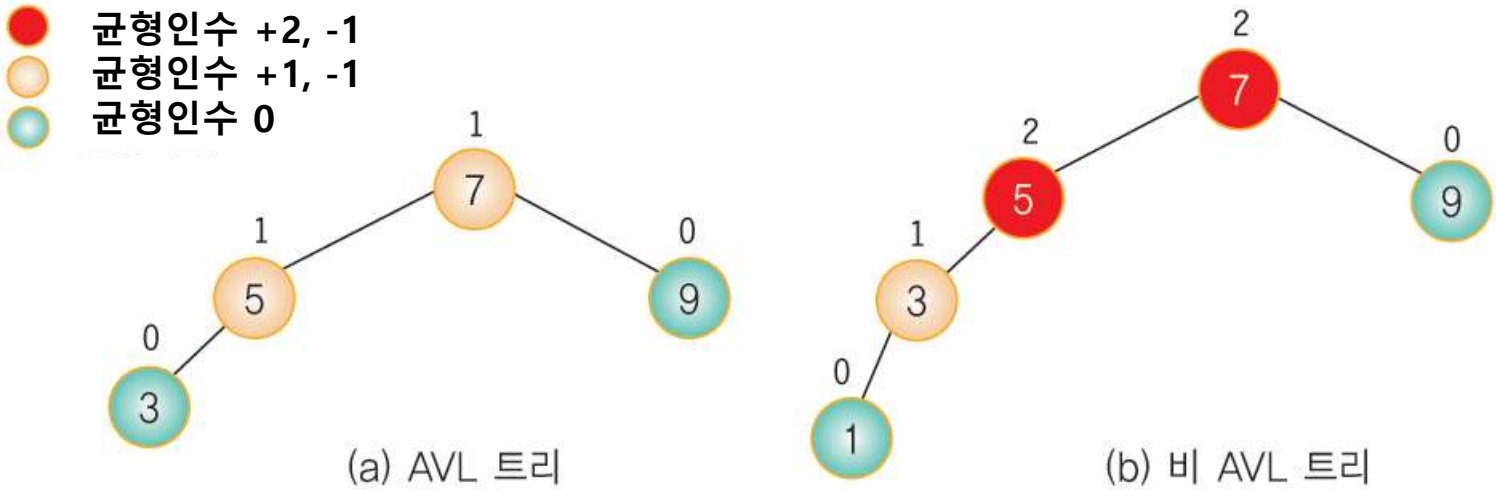
균형인수

- 균형 인수(Balance Factor, BF): 균형의 정도를 표현하는 단위
- 균형 인수의 절대값이 크면 클수록 트리의 균형이 무너진 상태
  ※ 균형 인수 값 = 왼쪽 서브 트리의 높이 - 오른쪽 서브 트리의 높이
- AVL 트리는 균형 인수의 절대값이 '2' 이상인 경우에 위치 재조정을 진행



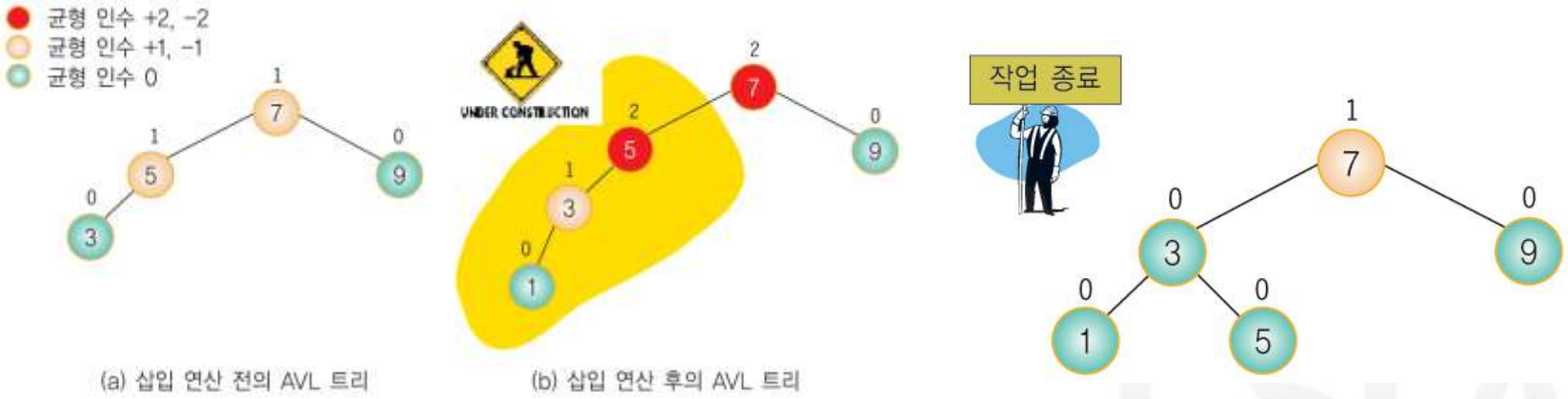
AVL 트리

- Adelson-Velskii와 Landis에 의해 1962년에 제안된 트리
- 모든 노드의 왼쪽과 오른쪽 서브트리의 높이 차가 1이하인 이진탐색트리
- 트리가 비균형 상태가 되면 노드들을 재배치하여 균형 상태 유지
- 모든 노드의 균형 인수가 ±1 이하이면 AVL 트리



AVL트리의 삽입연산

- 탐색연산: 이진탐색트리와 동일
- 삽입 연산과 삭제 연산 시 균형 상태가 깨질 수 있음
- 삽입 연산
  - 삽입 위치에서 루트까지의 경로에 있는 조상 노드들의 균형 인수 영향
  - 삽입 후에 불균형 상태로 변한 가장 가까운 조상 노드(균형 인수가 ±2가 된 가장 가까운 조상 노드)의 서브 트리들에 대하여 재배치
  - 삽입 노드부터 균형 인수가 ±2가 된 가장 가까운 조상 노드까지 회전



(a) 삽입 연산 전의 AVL 트리 (b) 삽입 연산 후의 AVL 트리

AVL트리의 삽입연산

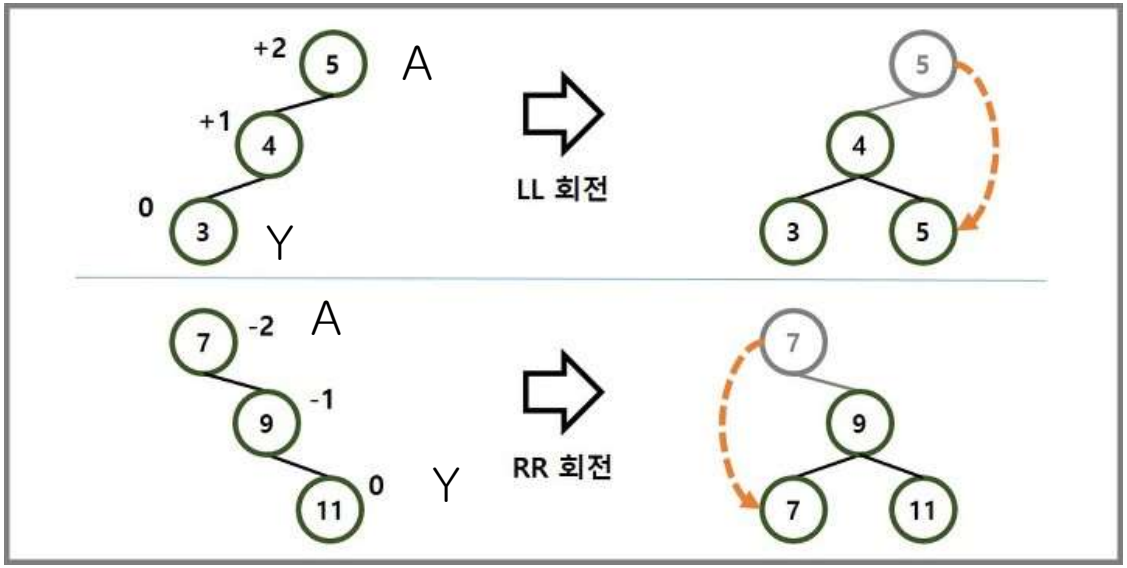
- **AVL 트리의 균형이 깨지는 4가지 경우(삽입된 노드 Y로부터 가장 가까우면서 균형 인수가 ±2가 된 조상 노드가 A라면)**
  - LL 타입: A의 왼쪽 자식의 왼쪽 서브 트리에 삽입
  - LR 타입: A의 왼쪽 자식의 오른쪽 서브 트리에 삽입
  - RR 타입: A의 오른쪽 자식의 오른쪽 서브 트리에 삽입
  - RL 타입: A의 오른쪽 자식의 왼쪽 서브 트리에 삽입

- **각 타입별 재균형 방법**
  - LL 회전: A부터 Y까지의 경로상 노드의 오른쪽 회전
  - LR 회전: A부터 Y까지의 경로상 노드의 왼쪽-오른쪽 회전
  - RR 회전: A부터 Y까지의 경로상 노드의 왼쪽 회전
  - RL 회전: A부터 Y까지의 경로상 노드의 오른쪽-왼쪽 회전

AVL 트리 단순회전 알고리즘

rotate_LL(A):
균형인수가 +-1을 초과하고, 새노드 Y는 A의 왼쪽 자식의 왼쪽 서브트리에 삽입되어있다.

rotate_RR(A):
균형인수가 +-1을 초과하고, 새노드 Y는 A의 오른쪽 자식의 오른쪽 서브트리에 삽입되어있다.

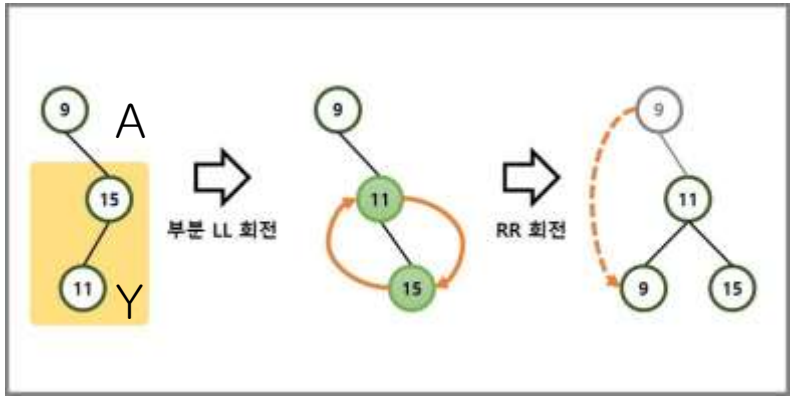


- 재균형 방법
  LL 회전: A부터 Y까지의 경로상 노드의 오른쪽 회전
  RR 회전: A부터 Y까지의 경로상 노드의 왼쪽 회전

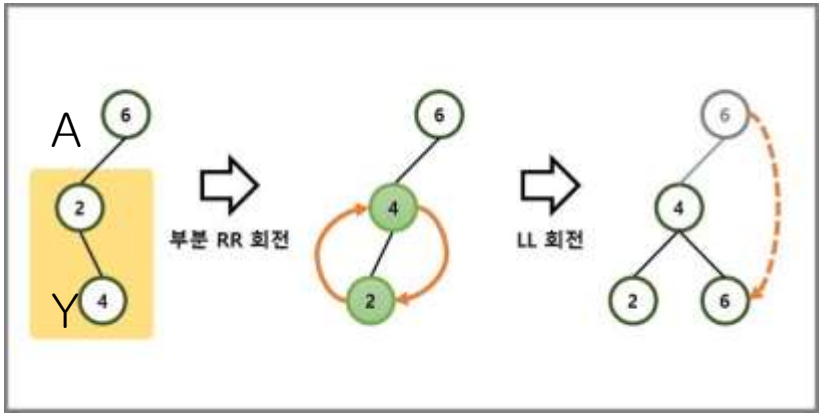
AVL 트리 이중회전 알고리즘

rotate_RL(A): 균형인수가 +-1을 초과하고, Y는 A의 오른쪽 자식의 왼쪽 서브트리에 삽입되있다.
rotate_right(B)
rotate_left(A)

rotate_LR(A): 균형인수가 +-1을 초과하고, Y는 A의 왼쪽 자식의 오른쪽 서브트리에 삽입되어있다.
rotate_left(B)
rotate_right(A)



rotate RL

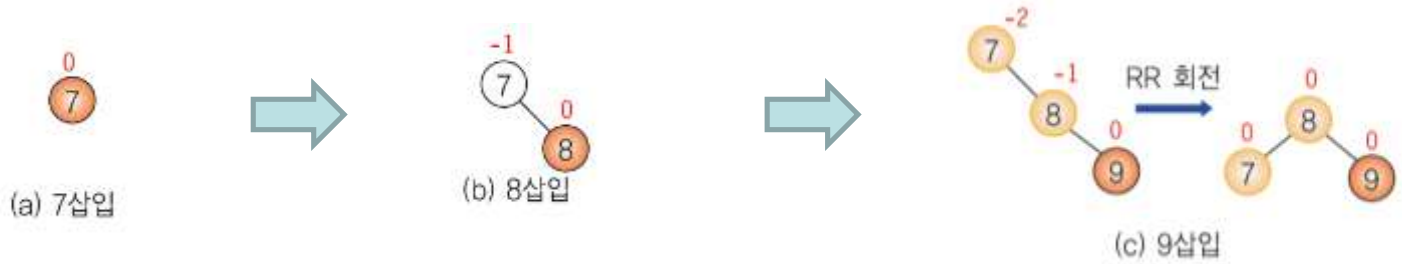


rotate LR

- 재균형 방법
  RL 회전: A부터 Y까지의 경로상 노드의 오른쪽-왼쪽 회전
  LR 회전: A부터 Y까지의 경로상 노드의 왼쪽-오른쪽 회전

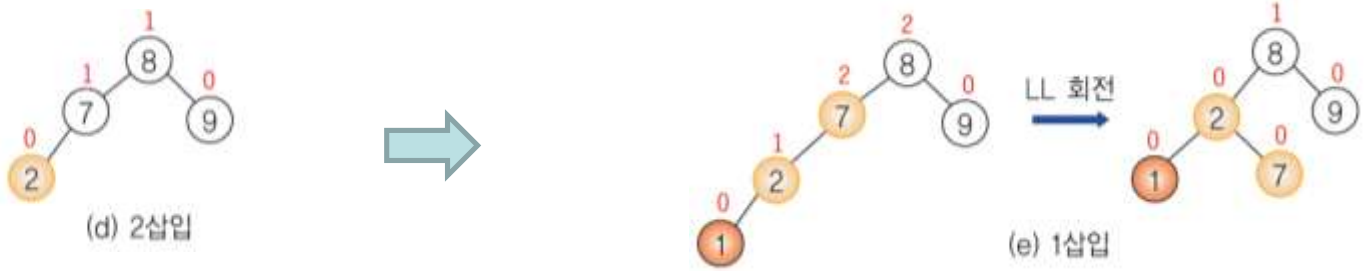
AVL 트리 삽입 예제

- **insert(7,8,9,**2,1,5,3,6,4**)**
  - 삽입된 노드는 빨간색
  - 위치가 변경되는 노드는 주황색



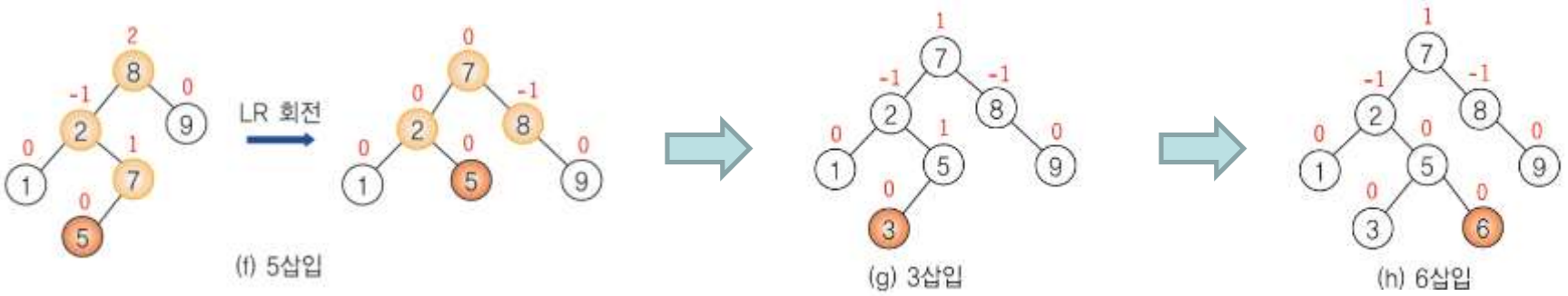
AVL 트리 삽입 예제

- **insert(7,8,9,2,1**,5,3,6,4)
  - 삽입된 노드는 빨간색
  - 위치가 변경되는 노드는 주황색



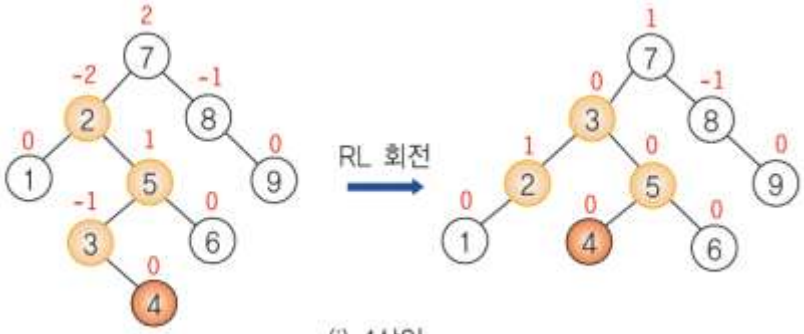
AVL 트리 삽입 예제

- **insert(7,8,9,2,1,5,3,6,4)**
  - 삽입된 노드는 빨간색
  - 위치가 변경되는 노드는 주황색



AVL 트리 삽입 예제

- **insert(7,8,9,2,1,5,3,6,4)**
  - 삽입된 노드는 빨간색
  - 위치가 변경되는 노드는 주황색



(i) 4삽입

AVL 트리 구현(개인실습 #1)

- 구조체

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

struct avl_node   {
    struct avl_node *left_child, *right_child;  /* Subtrees. */
    int data;                   /* Pointer to data. */
};
```

AVL 트리 구현

– 출력 함수

```
void display(struct avl_node *node)
{
        if (node != NULL) {
                printf("(");
                display(node->left_child);
                printf("%d", node->data);
                display(node->right_child);
                printf(")");
        }
}
```

AVL 트리 구현

- 노드의 균형인수 반환, 트리의 높이를 반환

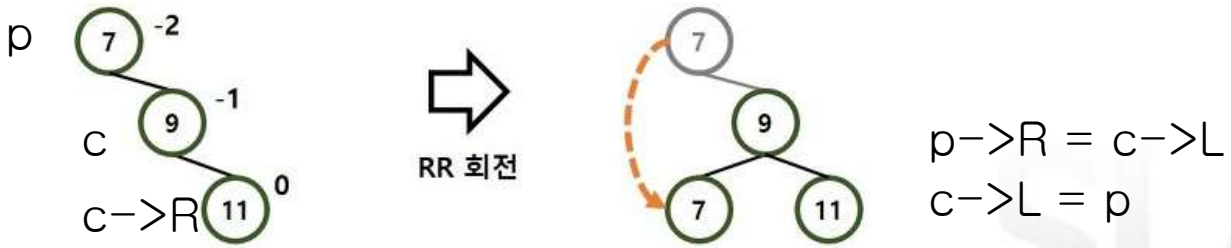
```
// 트리의 높이를 반환
int get_height(struct avl_node *node)
{
        int height=0;
        /* fill in the blank */
        return height;
}

// 노드의 균형인수를 반환
int get_height_diff(struct avl_node *node)
{
        int height_diff =0;
        if( node == NULL ) return 0;
        /* fill in the blank */

}
```

AVL 트리 구현

- rotate_left
- rotate_right

```
struct avl_node* rotate_left(struct avl_node *parent)
{
        struct avl_node *child = parent->right_child;
         /* fill in the blank */
        return child;
}
```

HINT:



AVL 트리 구현

- rotate_left_right
- rotate_right_left

```
struct avl_node* rotate_left_right(struct avl_node *parent)
{
        struct avl_node *child = parent->left_child;
         /* fill in the blank */
}
```

AVL 트리 구현

- 트리를 균형트리로 만듬

```
struct avl_node* rebalance(struct avl_node **node)
{
        int height_diff = get_height_diff(*node);
        /*                      */

        return *node;
}
```

AVL 트리 구현

- 삽입 함수

```
struct avl_node* avl_add(struct avl_node** root, int new_key)
{
/*          중복 키 허용 x
            BST와 삽입과정은 비슷          */

}
```

AVL 트리 구현

– 메인함수

```
struct avl_node *root; //전역 선언함

void main()
{
        avl_add(&root, 7);
        avl_add(&root, 8);
        avl_add(&root, 9);
        avl_add(&root, 2);
        avl_add(&root, 1);

        avl_add(&root, 5);
        avl_add(&root, 3);
        avl_add(&root, 6);
        avl_add(&root, 4);
        display(root);
}
```

```
((((1)2)3((4)5(6)))7(8(9)))
```

제출

- **제출**
  - 개인실습 (#1)
    - 오늘 자정까지 제출 (~ 2020/5/8 23:59)
    - AVL Tree (insert)