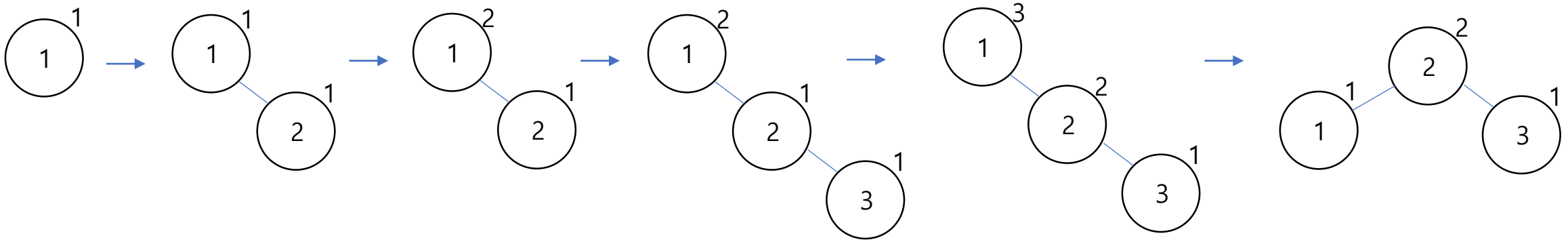
TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 문한나
mhn97@naver.com

AVL Tree - 복습

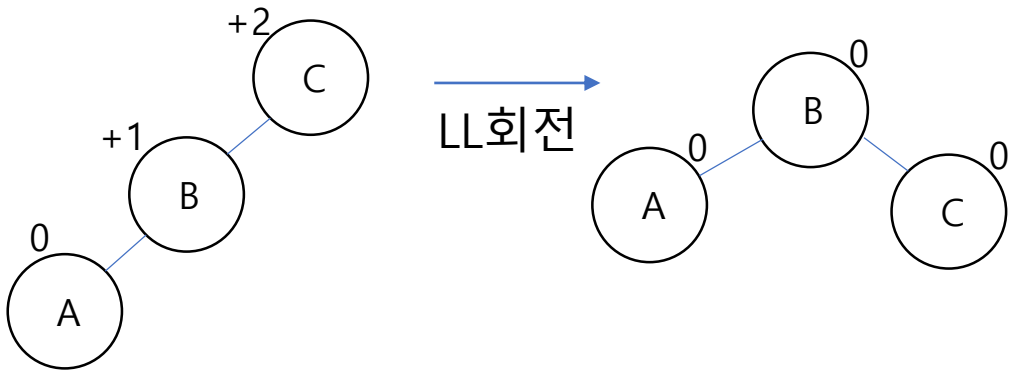
AVL트리 : 한 노드를 중심으로 좌우 종속 트리의 높이 차가 1 이하인 균형 잡힌 트리
AVL트리는 이진 트리의 삽입, 삭제를 계속할 때 어느 한 방향으로 치우치거나, 높이 차이로 인해서
수행시간이 증가되는 것을 막기 위해 균형을 유지하도록 한 것이다

만약 삽입, 삭제 후 2이상의 균형 인수차이가 나게 된다면 삽입,삭제 된 노드에서 가장 가까운 조상 노드의
서브 트리들에 대해 다시 균형을 맞추기 위한 재배치를 실시한다.



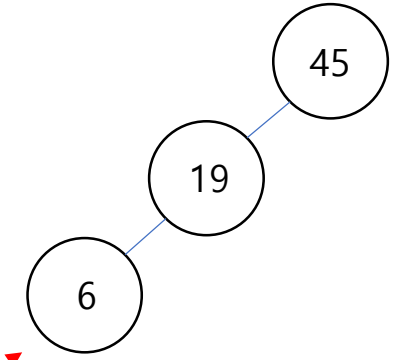
AVL트리가 깨지는 총 4가지 경우(N : 새로 삽입된 노드, A : N의 조상으로 균형인자가 ±2 되는 노드)

LL : N 가 A의 왼쪽의 왼쪽 부속 트리에 삽입됐을 때

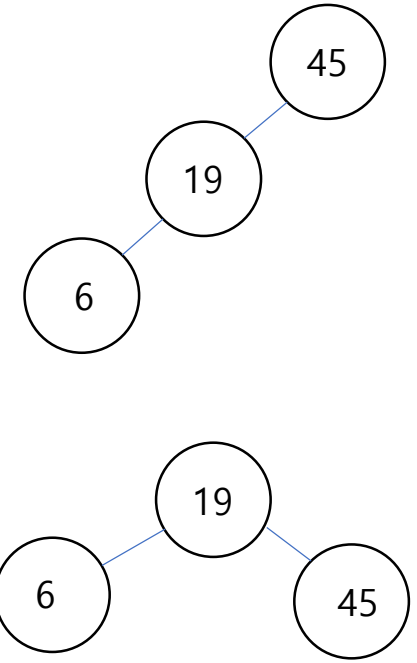


오른쪽 회전을 시키면 다시 균형 회복

```
.mhn@mhn-900X3L:~/my_proj/c/12_s$ ./a.out
arr[0] = 51
arr[1] = 75
arr[2] = 52
arr[3] = 83
arr[4] = 78
arr[5] = 63
arr[6] = 19
arr[7] = 67
arr[8] = 45
arr[9] = 99
arr[10] = 6
arr[11] = 96
arr[12] = 60
arr[13] = 9
arr[14] = 7
Insert Rotation!
data = 52
RL Rotation
Insert Rotation!
data = 78
RL Rotation
Insert Rotation!
data = 63
RL Rotation
Insert Rotation!
data = 45
LR Rotation
Insert Rotation!
data = 99
RR Rotation
Insert Rotation!
data = 6
LL Rotation
Insert Rotation!
data = 9
LR Rotation
Insert Rotation!
data = 7
LL Rotation
data = 52, lev = 5, left = 9, right = 75
data = 9, lev = 3, left = 6, right = 45
data = 6, lev = 2, left = NULL, right = 7
data = 7, lev = 1, left = NULL, right = NULL
data = 45, lev = 2, left = 19, right = 51
data = 19, lev = 1, left = NULL, right = NULL
data = 51, lev = 1, left = NULL, right = NULL
data = 75, lev = 4, left = 63, right = 83
data = 63, lev = 2, left = 60, right = 67
data = 60, lev = 1, left = NULL, right = NULL
data = 67, lev = 1, left = NULL, right = NULL
data = 83, lev = 3, left = 78, right = 99
data = 78, lev = 1, left = NULL, right = NULL
data = 99, lev = 2, left = 96, right = NULL
data = 96, lev = 1, left = NULL, right = NULL
```

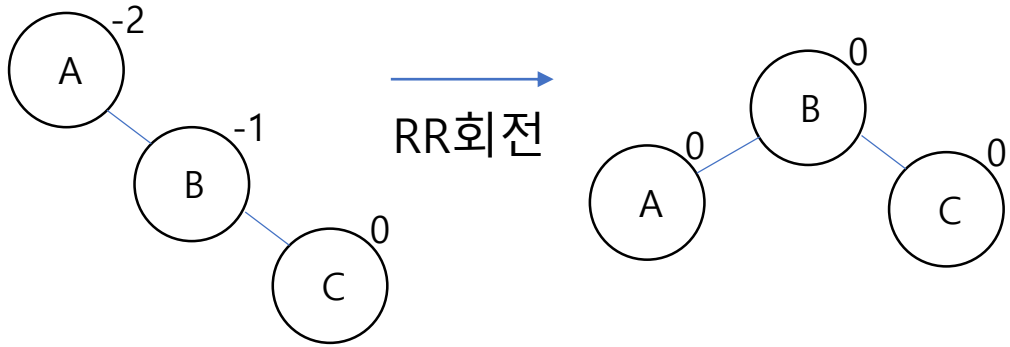



```
}

avl *rr_rot(avl *parent, avl *child)
{
        parent->right = child->left;
        child->left = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

avl *ll_rot(avl *parent, avl *child)
{
        parent->left = child->right;
        child->right = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

avl *rl_rot(avl *parent, avl *child)
{
        child = ll_rot(child, child->left);
        return rr_rot(parent, child);
}

avl *lr_rot(avl *parent, avl *child)
{
        child = rr_rot(child, child->right);
        return ll_rot(parent, child);
}

//void rotation(avl *root, int ret)
avl *rotation(avl *root, int ret)
{
        switch(ret)
        {
                case RL:
                        printf("RL Rotation\n");
                        return rl_rot(root, root->right);
                case RR:
                        printf("RR Rotation\n");
                        return rr_rot(root, root->right);
                case LR:
                        printf("LR Rotation\n");
                        return lr_rot(root, root->left);
                case LL:
                        printf("LL Rotation\n");
                        return ll_rot(root, root->left);
        }
}
```



AVL트리가 깨지는 총 4가지 경우(N : 새로 삽입된 노드, A : N의 조상으로 균형인자가 ±2 되는 노드)

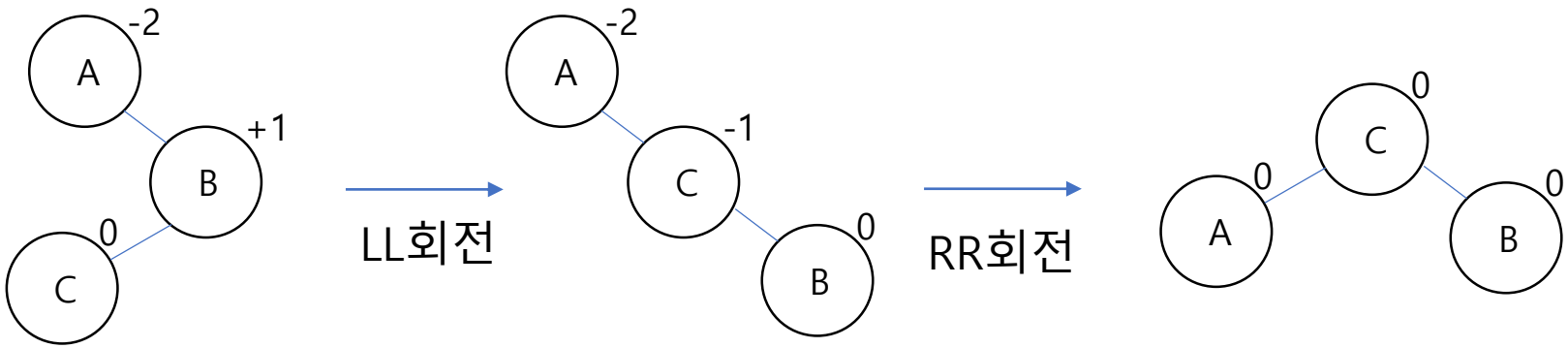
RR : N 가 A의 오른쪽의 오른쪽 부속 트리에 삽입됐을 때



왼쪽 회전을 시키면 다시 균형 회복

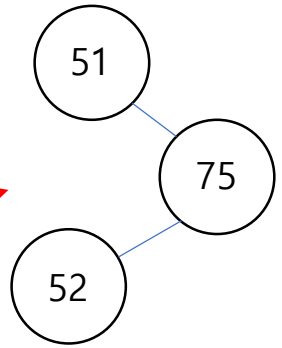
AVL트리가 깨지는 총 4가지 경우(N : 새로 삽입된 노드, A : N의 조상으로 균형인자가 ±2 되는 노드)

RL : N 가 A의 오른쪽의 왼쪽 부속 트리에 삽입됐을 때

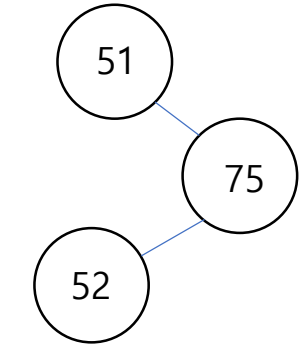


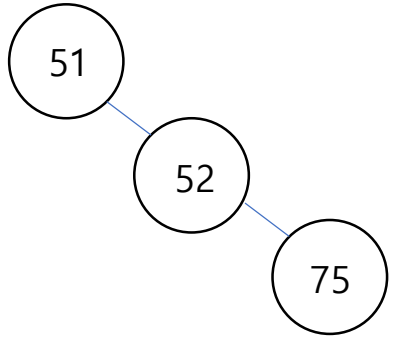
오른쪽 회전 후 왼쪽 회전을 시키면 다시 균형 회복

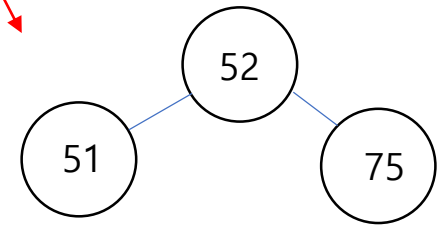
```
.mhn@mhn-900X3L:~/my_proj/c/12_s$ ./a.out
arr[0] = 51
arr[1] = 75
arr[2] = 52
arr[3] = 83
arr[4] = 78
arr[5] = 63
arr[6] = 19
arr[7] = 67
arr[8] = 45
arr[9] = 99
arr[10] = 6
arr[11] = 96
arr[12] = 60
arr[13] = 9
arr[14] = 7
Insert Rotation!
data = 52
RL Rotation
Insert Rotation!
data = 78
RL Rotation
Insert Rotation!
data = 63
RL Rotation
Insert Rotation!
data = 45
LR Rotation
Insert Rotation!
data = 99
RR Rotation
Insert Rotation!
data = 6
LL Rotation
Insert Rotation!
data = 9
LR Rotation
Insert Rotation!
data = 7
LL Rotation
data = 52, lev = 5, left = 9, right = 75
data = 9, lev = 3, left = 6, right = 45
data = 6, lev = 2, left = NULL, right = 7
data = 7, lev = 1, left = NULL, right = NULL
data = 45, lev = 2, left = 19, right = 51
data = 19, lev = 1, left = NULL, right = NULL
data = 51, lev = 1, left = NULL, right = NULL
data = 75, lev = 4, left = 63, right = 83
data = 63, lev = 2, left = 60, right = 67
data = 60, lev = 1, left = NULL, right = NULL
data = 67, lev = 1, left = NULL, right = NULL
data = 83, lev = 3, left = 78, right = 99
data = 78, lev = 1, left = NULL, right = NULL
data = 99, lev = 2, left = 96, right = NULL
data = 96, lev = 1, left = NULL, right = NULL
```




```
}

avl *rr_rot(avl *parent, avl *child)
{
        parent->right = child->left;
        child->left = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

avl *ll_rot(avl *parent, avl *child)
{
        parent->left = child->right;
        child->right = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

avl *rl_rot(avl *parent, avl *child)
{
        child = ll_rot(child, child->left);
        return rr_rot(parent, child);
}

avl *lr_rot(avl *parent, avl *child)
{
        child = rr_rot(child, child->right);
        return ll_rot(parent, child);
}

//void rotation(avl *root, int ret)
avl *rotation(avl *root, int ret)
{
        switch(ret)
        {
                case RL:
                        printf("RL Rotation\n");
                        return rl_rot(root, root->right);
                case RR:
                        printf("RR Rotation\n");
                        return rr_rot(root, root->right);
                case LR:
                        printf("LR Rotation\n");
                        return lr_rot(root, root->left);
                case LL:
                        printf("LL Rotation\n");
                        return ll_rot(root, root->left);
        }
}
```

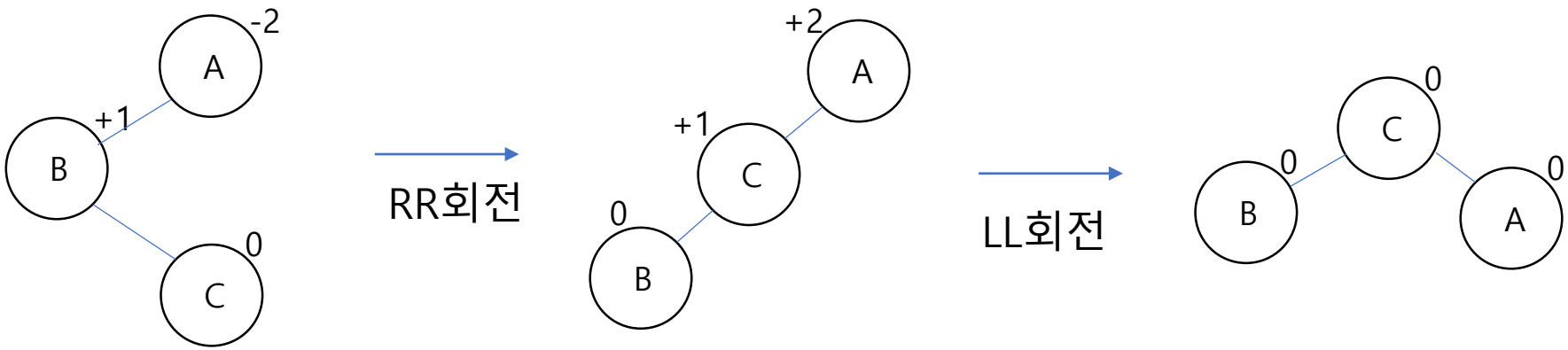






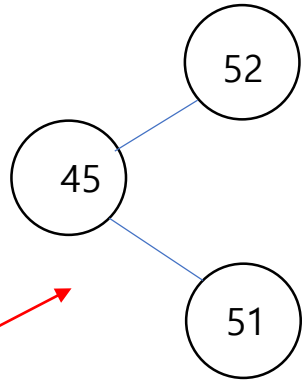
AVL트리가 깨지는 총 4가지 경우(N : 새로 삽입된 노드, A : N의 조상으로 균형인자가 ±2 되는 노드)

LR : N 가 A의 왼쪽의 오른쪽 부속 트리에 삽입됐을 때

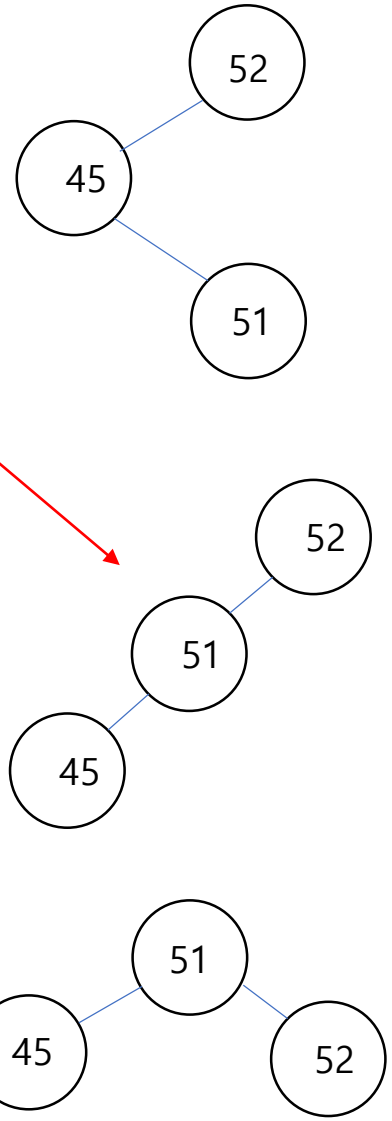


왼쪽 회전 후 오른쪽 회전을 시키면 다시 균형 회복

```
.mhn@mhn-900X3L:~/my_proj/c/12_s$ ./a.out
arr[0] = 51
arr[1] = 75
arr[2] = 52
arr[3] = 83
arr[4] = 78
arr[5] = 63
arr[6] = 19
arr[7] = 67
arr[8] = 45
arr[9] = 99
arr[10] = 6
arr[11] = 96
arr[12] = 60
arr[13] = 9
arr[14] = 7
Insert Rotation!
data = 52
RL Rotation
Insert Rotation!
data = 78
RL Rotation
Insert Rotation!
data = 63
RL Rotation
Insert Rotation!
data = 45
LR Rotation
Insert Rotation!
data = 99
RR Rotation
Insert Rotation!
data = 6
LL Rotation
Insert Rotation!
data = 9
LR Rotation
Insert Rotation!
data = 7
LL Rotation
data = 52, lev = 5, left = 9, right = 75
data = 9, lev = 3, left = 6, right = 45
data = 6, lev = 2, left = NULL, right = 7
data = 7, lev = 1, left = NULL, right = NULL
data = 45, lev = 2, left = 19, right = 51
data = 19, lev = 1, left = NULL, right = NULL
data = 51, lev = 1, left = NULL, right = NULL
data = 75, lev = 4, left = 63, right = 83
data = 63, lev = 2, left = 60, right = 67
data = 60, lev = 1, left = NULL, right = NULL
data = 67, lev = 1, left = NULL, right = NULL
data = 83, lev = 3, left = 78, right = 99
data = 78, lev = 1, left = NULL, right = NULL
data = 99, lev = 2, left = 96, right = NULL
data = 96, lev = 1, left = NULL, right = NULL
```




```
}

avl *rr_rot(avl *parent, avl *child)
{
        parent->right = child->left;
        child->left = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

avl *ll_rot(avl *parent, avl *child)
{
        parent->left = child->right;
        child->right = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

avl *rl_rot(avl *parent, avl *child)
{
        child = ll_rot(child, child->left);
        return rr_rot(parent, child);
}

avl *lr_rot(avl *parent, avl *child)
{
        child = rr_rot(child, child->right);
        return ll_rot(parent, child);
}

//void rotation(avl *root, int ret)
avl *rotation(avl *root, int ret)
{
        switch(ret)
        {
                case RL:
                        printf("RL Rotation\n");
                        return rl_rot(root, root->right);
                case RR:
                        printf("RR Rotation\n");
                        return rr_rot(root, root->right);
                case LR:
                        printf("LR Rotation\n");
                        return lr_rot(root, root->left);
                case LL:
                        printf("LL Rotation\n");
                        return ll_rot(root, root->left);
        }
}
```



Red-Black Tree

특징 1 : 루트노드는 항상 검정이다.
특징 2 : 잎사귀 노드는 어디를 가던지 거치는 검정색의 개수가 모두 같다.
특징 3 : 빨강이 연속해서 두개 오면 회전하거나 색상을 바꾼다.
특징 4 : 현재 기준점에서 부모노드와 삼촌의 색상이 같으면 색상만 변경함.
-> 할아버지가 빨간색이 되고 자식들은 검정색이 됨
특징 5 : 3번규칙을 만족하는데 4번이 만족되지 않으면 회전한다.