윤성우의 열혈 자료구조

: C언어를 이용한 자료구조 학습서

Chapter 12. 탐색2

Introduction To Data Structures Using C

Chapter 12-1:

균형 잡힌 이진 탐색 트리: AVL 트리의 이해

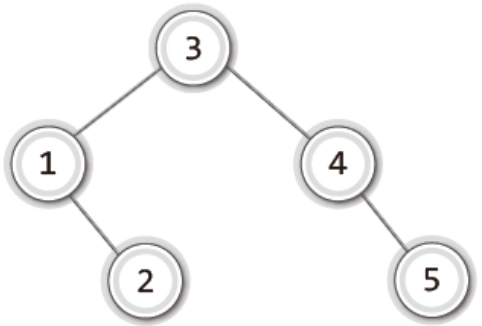
이진 탐색 트리의 문제점과 AVL 트리

1부터 5까지 순서대로 저장이 이뤄진 경우!

이진 탐색 트리의 탐색 연산은 $O(log_2 n)$의 시간 복잡도를 보인다.

그러나 균형이 맞지 않을수록 $O(n)$에 가까운 시간 복잡도를 보인다.

3이 제일 먼저 저장된 경우!



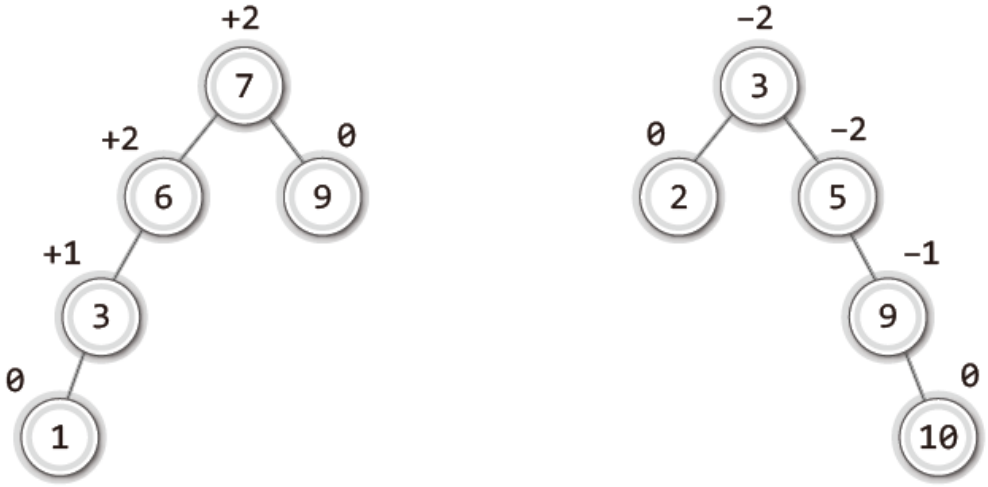
약간의 순서 변화로 균형이 잡혔다!

이진 탐색 트리의 균형 문제를 해결한 트리

- AVL 트리
- 2-3-4 트리
- Red-Black 트리
- 등등...

자동으로 균형 잡는 AVL 트리와 균형 인수

균형 인수 = 왼쪽 서브 트리의 높이 – 오른쪽 서브 트리의 높이



균형 인수의 절댓값이 2 이상인 경우 리밸런싱 진행!

AVL 트리는 균형 인수(Balance Factor)를 기준으로 트리의 균형을 잡기 위한 재조정(리밸런싱)의 진행 시기를 결정한다.

리밸런싱이 필요한 첫 번째 상태와 LL회전

▶ [그림 12-4: LL회전의 방법과 그 결과]

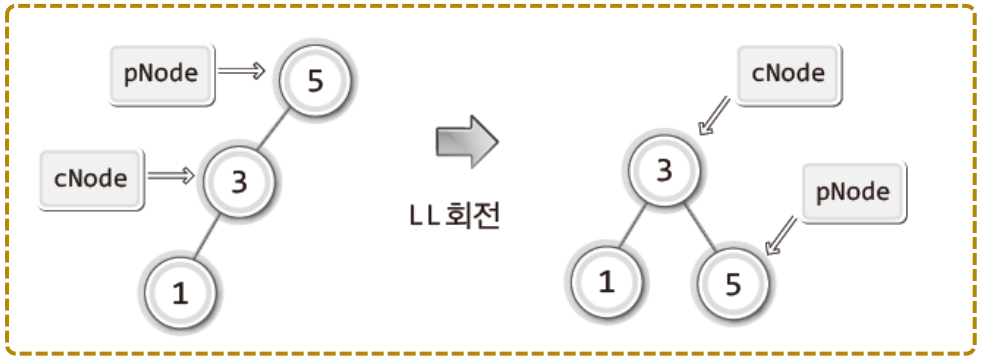
LL상태

"5가 저장된 노드의 **왼쪽(Left)**에 3이 저장된 자식 노드가 하나 존재하고, 그 자식 노드의 **왼쪽(Left)**에 1이 저장된 자식 노드가 또 하나 존재한다."

이러한 LL상태를 균형 잡기 위해서 LL회전을 진행한다.

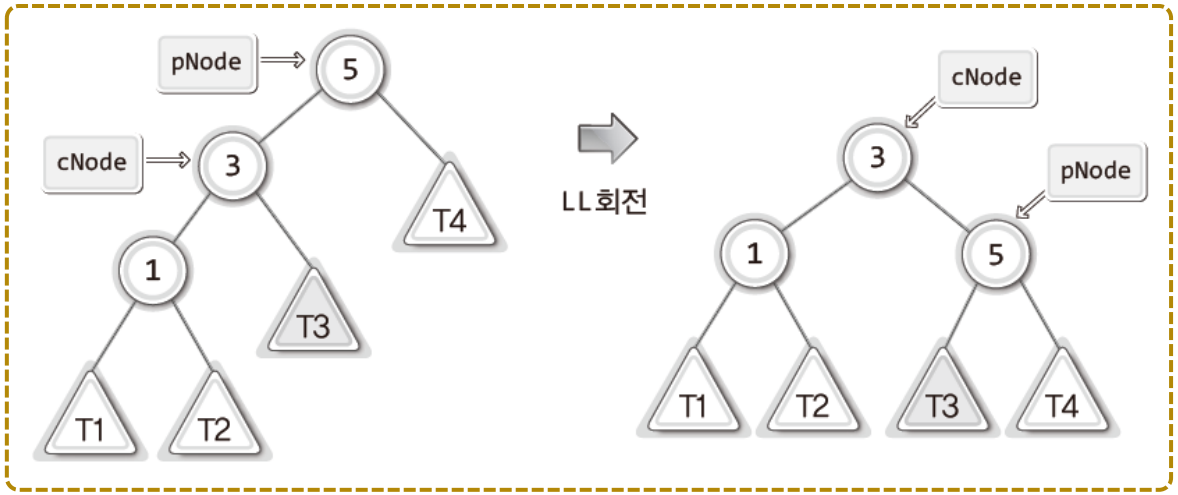
LL상태를 균형 잡기 위한 LL회전

단순한 예



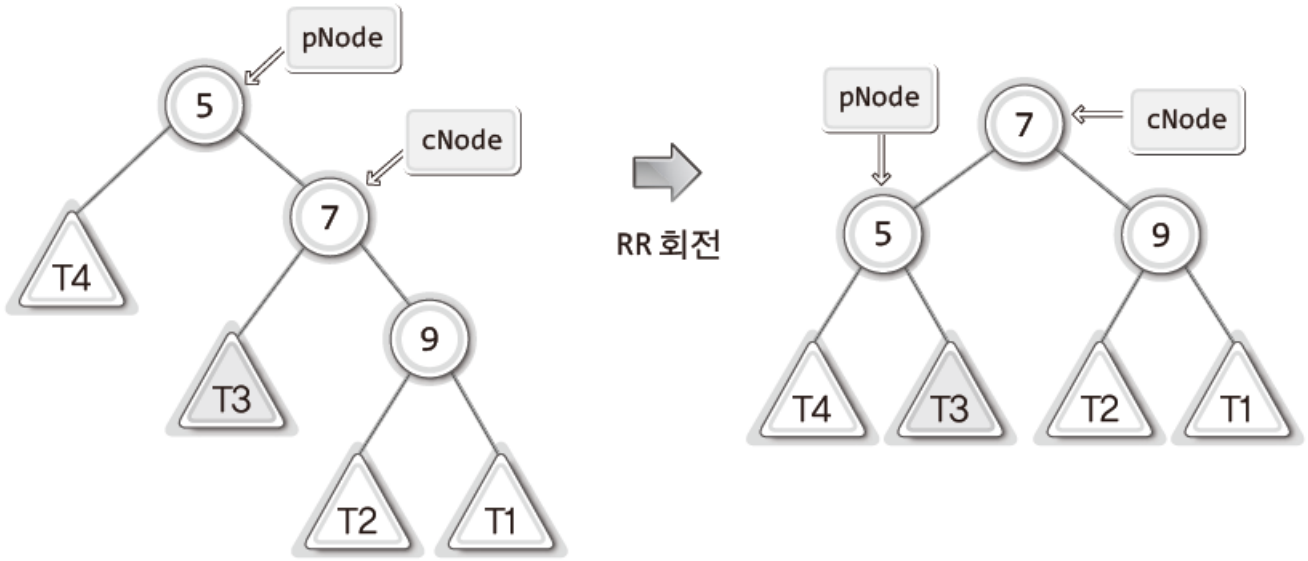

```
ChangeRightSubTree(cNode, pNode);
```

단순한 예의 일반화




```
ChangeLeftSubTree(pNode, GetRightSubTree(cNode));
ChangeRightSubTree(cNode, pNode);
```

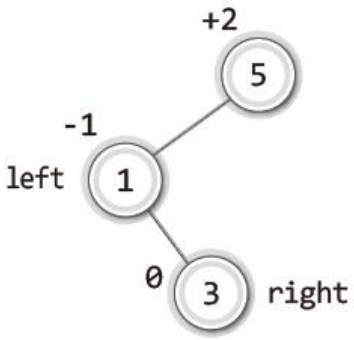
RR상태와 RR회전




```
ChangeRightSubTree(pNode, GetLeftSubTree(cNode));
ChangeLeftSubTree(cNode, pNode);
```

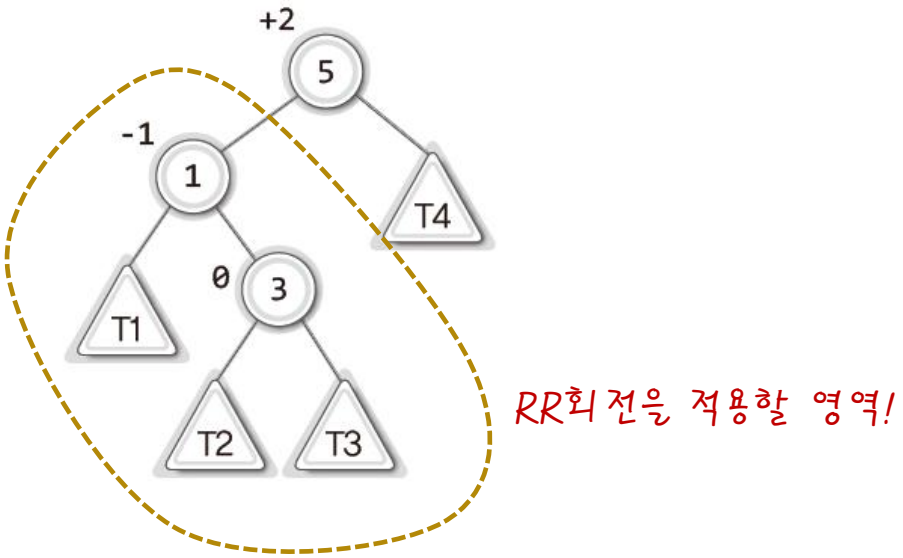
LR상태

LL상태 그리고 RR상태와 같이 한 번의 회전으로 균형을 잡을 수 없다. 따라서 LR 상태는 한 번의 회전으로 균형이 잡히는 LL상태 또는 RR상태가 되도록 하는 것이 우선이다!



간단한 LR상태

일반화



LR상태는 RR회전을 통해서(RR회전의 부수적인 효과를 이용해서) LL상태가 되게 할 수 있다.

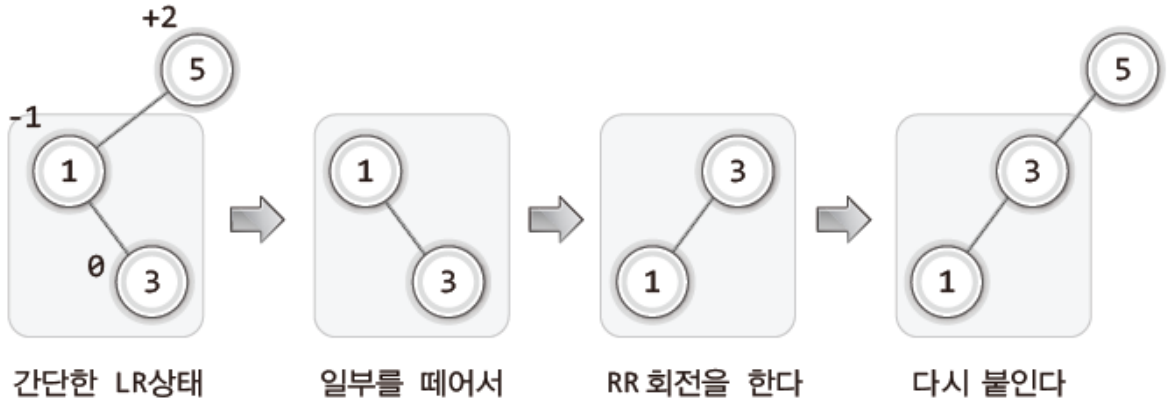
RR회전의 부수적인 효과

일반적인 RR회전

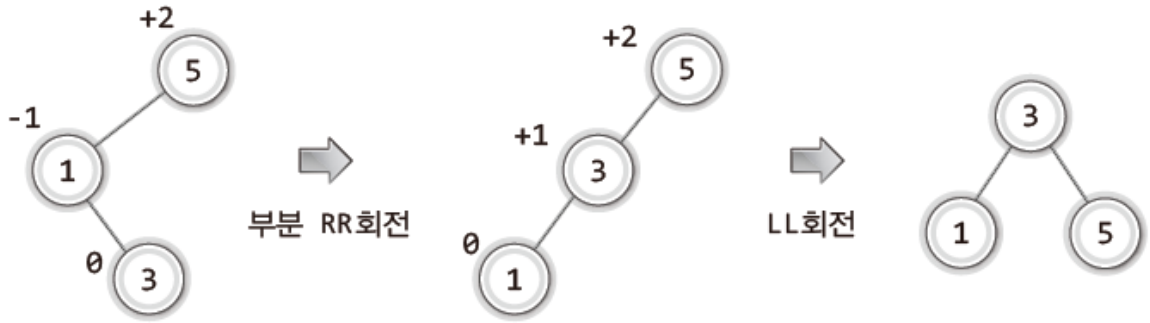
단말 노드가 NULL인 경우에도 RR 회전 가능하다!

부모 자식의 관계가 바뀌는 부수적인 효과,

LR회전



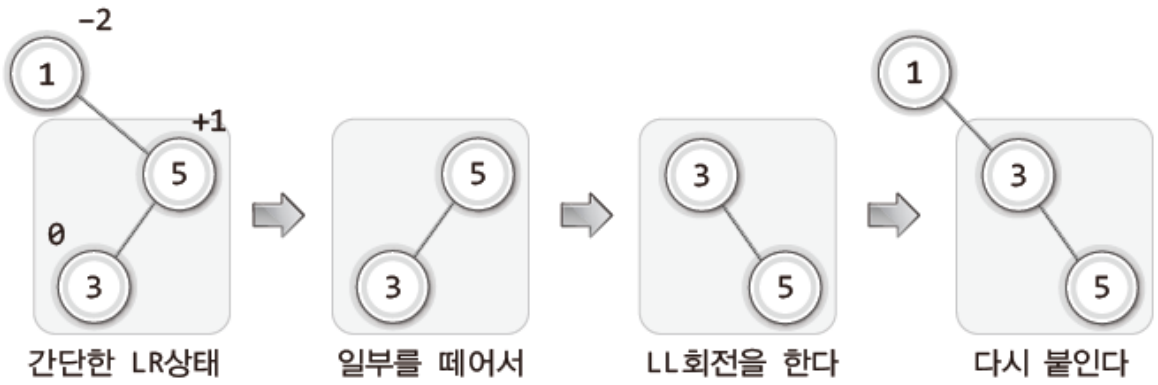
LR상태를 LL상태로



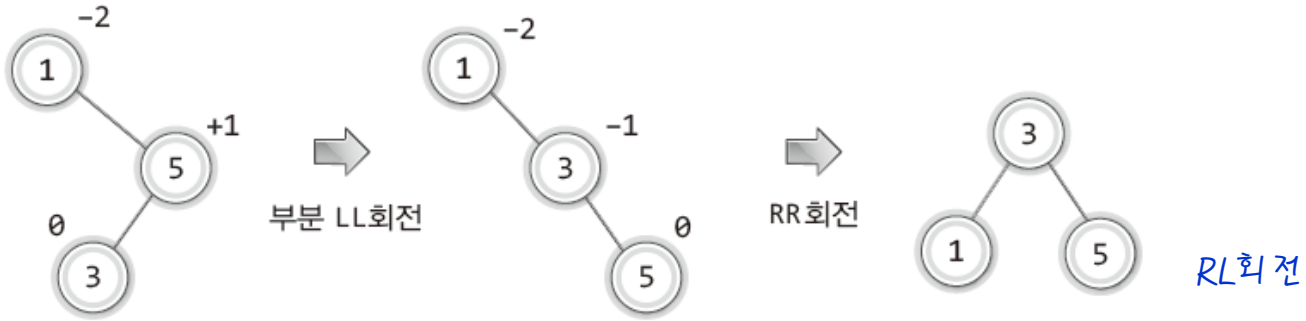
LR회전

RL상태와 RL회전

LR상태 LR회전, 그리고 RL상태 RL회전은 방향에서만 차이를 보인다.



RL상태는 RR상태로



RL회전

Chapter 12-2:

AVL 트리의 구현

AVL 트리를 어떻게 구현할 것인가?

활용할 파일들

- BinaryTree3.h — 이진 트리의 헤더파일
- BinaryTree3.c — 이진 트리를 구성하는데 필요한 도구들의 모임
- BinarySearchTree2.h — 이진 탐색 트리의 헤더파일
- BinarySearchTree2.c — 이진 탐색 트리의 구현

AVL 트리도 이진 탐색 트리의 일종이므로 앞서 구현한 이진 탐색 트리를 기반으로 구현한다.

BinarySearchTree2.c에 리밸런싱 기능을 추가하여, 파일의 이름을 BinarySearchTree3.c로 변경하자!

단 다음 두 파일을 추가하여 리밸런싱 도구를 정의하기로 하겠다.

새로 작성할 파일들

- AVLRebalance.h — 리밸런싱 관련 함수들의 선언
- AVLRebalance.c — 리밸런싱 관련 함수들의 정의

AVL 트리 구현을 위한 확장 포인트

확장할 함수들

- BSTInsert 함수 트리에 노드를 추가
- BSTRemove 함수 트리에서 노드를 제거

균형은 노드의 삽입과 삭제의 순간에 깨지게 된다.

확장의 형태

```
void BSTInsert(BTreeNode ** pRoot, BSTData data)
{
    . . . .
    Rebalance(pRoot);     // 노드 추가 후 리밸런싱!
}

BTreeNode * BSTRemove(BTreeNode ** pRoot, BSTData target)
{
    . . . .
    Rebalance(pRoot);     // 노드 제거 후 리밸런싱!
    return dNode;
}
```

리밸런싱 도구: 균형을 이루고 있는가?

```
// 두 서브 트리의 '높이의 차(균형 인수)'를 반환
int GetHeightDiff(BTreeNode * bst)
{
     int lsh;       // left sub tree height
     int rsh;       // right sub tree height

     if(bst == NULL)
         return 0;

     lsh = GetHeight(GetLeftSubTree(bst));
     rsh = GetHeight(GetRightSubTree(bst));
     return lsh - rsh;     // 균형 인수 계산결과 반환
}
```

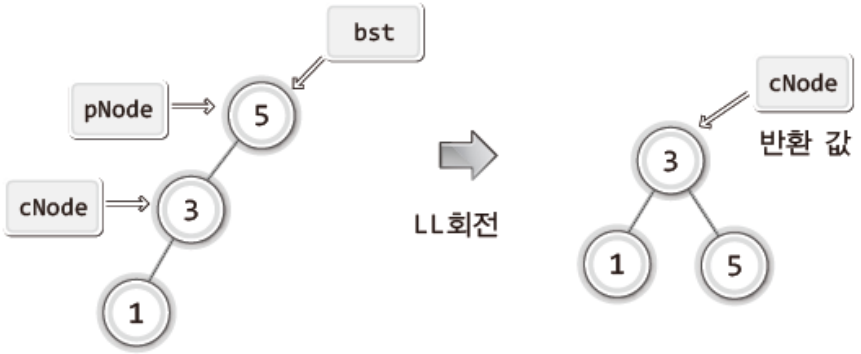
모든 경로의 높이를 비교하기 위한 재귀적 구성

```
// 트리의 높이를 계산하여 반환
int GetHeight(BTreeNode * bst)
{
     int leftH;     // left height
     int rightH;    // right height

     if(bst == NULL)
         return 0;

     leftH = GetHeight(GetLeftSubTree(bst));
     rightH = GetHeight(GetRightSubTree(bst));

     // 큰 값의 높이를 반환한다.
     if(leftH > rightH)
         return leftH + 1;
     else
         return rightH + 1;
}
```

리밸런싱 도구: LL회전

```
BTreeNode * RotateLL(BTreeNode * bst)      // LL회전을 담당하는 함수
{
    BTreeNode * pNode;        // parent node
    BTreeNode * cNode;        // child node

    // pNode와 cNode가 LL회전을 위해 적절한 위치를 가리키게 한다.
    pNode = bst;
    cNode = GetLeftSubTree(pNode);

    // 실제 LL회전을 담당하는 두 개의 문장
    ChangeLeftSubTree(pNode, GetRightSubTree(cNode));
    ChangeRightSubTree(cNode, pNode);

    // LL회전으로 인해서 변경된 루트 노드의 주소 값 반환
    return cNode;
}
```

회전 후 루트 노드가 변경되기 때문에
새로운 루트 노드의 주소 값을 반환해준다.



리밸런싱 도구: RR회전

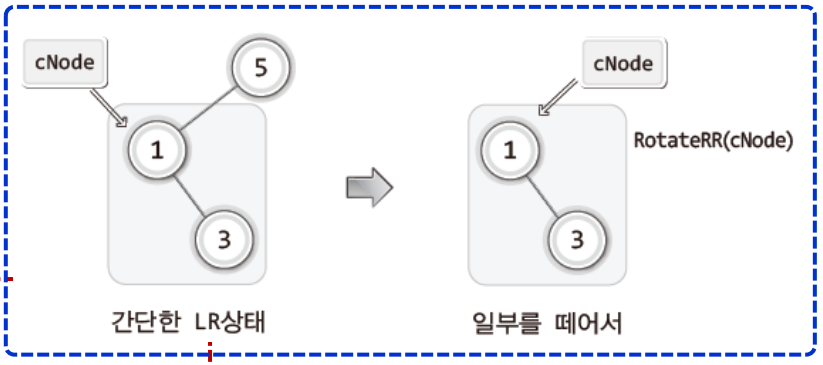
```
BTreeNode * RotateRR(BTreeNode * bst)    // RR회전을 담당하는 함수
{
    BTreeNode * pNode;        // parent node
    BTreeNode * cNode;        // child node

    // pNode와 cNode가 RR회전을 위해 적절한 위치를 가리키게 한다.
    pNode = bst;
    cNode = GetRightSubTree(pNode);

    // 실제 RR회전을 담당하는 두 개의 문장
    ChangeRightSubTree(pNode, GetLeftSubTree(cNode));
    ChangeLeftSubTree(cNode, pNode);

    // RR회전으로 인해서 변경된 루트 노드의 주소 값 반환
    return cNode;
}
```

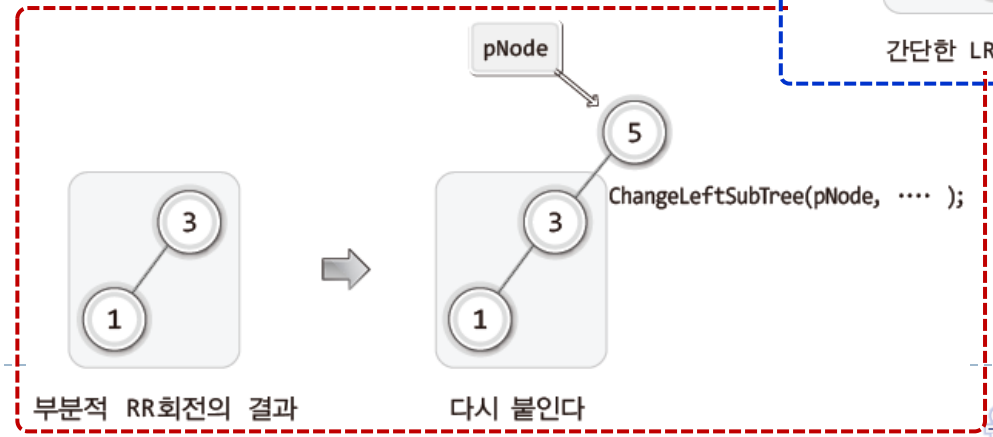
방향에 있어서만 차이를 보인다.

```
// 실제 LL회전을 담당하는 두 개의 문장
ChangeLeftSubTree(pNode, GetRightSubTree(cNode));
ChangeRightSubTree(cNode, pNode);
```

리밸런싱 도구: LR회전

```
// 부분적 RR회전에 이어서 LL회전을 진행
BTreeNode * RotateLR(BTreeNode * bst)      // LR회전을 담당하는 함수
{
    BTreeNode * pNode;        // parent node
    BTreeNode * cNode;        // child node

    // pNode와 cNode가 LR회전을 위해 적절한 위치를 가리키게 한다.
    pNode = bst;
    cNode = GetLeftSubTree(pNode);

    // 실제 LR회전을 담당하는 두 개의 문장
    ChangeLeftSubTree(pNode, RotateRR(cNode));
    return RotateLL(pNode);
}
```





리밸런싱 도구: RL회전

```
// 부분적 LL회전에 이어서 RR회전을 진행
BTreeNode * RotateRL(BTreeNode * bst)
{
    BTreeNode * pNode;        // parent node
    BTreeNode * cNode;        // child node

    // pNode와 cNode가 RL회전을 위해 적절한 위치를 가리키게 한다.
    pNode = bst;
    cNode = GetRightSubTree(pNode);

    // 실제 RL회전을 담당하는 두 개의 문장
    ChangeRightSubTree(pNode, RotateLL(cNode));     // 부분적 LL회전
    return RotateRR(pNode);                         // RR회전
}
```

LR회전에서는 GetLeftSubTree 호출

방향에 있어서만 차이를 보인다.

```
// 실제 LR회전을 담당하는 두 개의 문장
ChangeLeftSubTree(pNode, RotateRR(cNode));      // 부분적 RR회전
return RotateLL(pNode);                         // LL회전
```

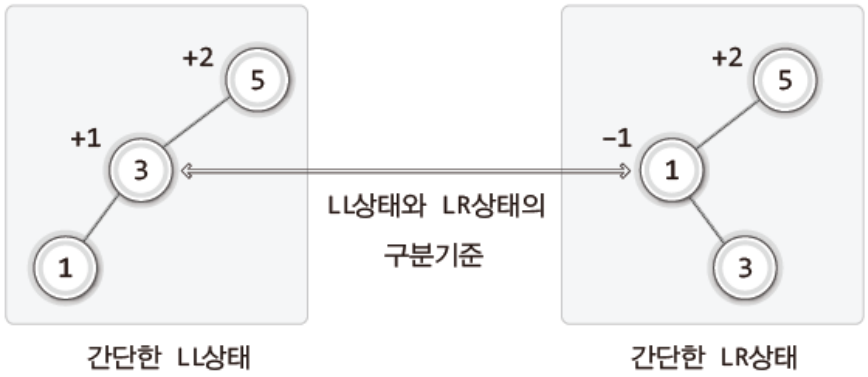
리밸런싱 도구: Rebalance 함수

```
BTreeNode * Rebalance(BTreeNode ** pRoot)
{
    int hDiff = GetHeightDiff(*pRoot);      // 균형 인수 계산

    // 균형 인수가 +2 이상이면 LL상태 또는 LR상태이다.
    if(hDiff > 1)     // 왼쪽 서브 트리 방향으로 높이가 2 이상 크다면,
    {
        if(GetHeightDiff(GetLeftSubTree(*pRoot)) > 0)
            *pRoot = RotateLL(*pRoot);
        else
            *pRoot = RotateLR(*pRoot);
    }

    // 균형 인수가 -2 이하이면 RR상태 또는 RL상태이다.
    if(hDiff < -1)        // 오른쪽 서브 트리 방향으로 2 이상 크다면,
    {
        if(GetHeightDiff(GetRightSubTree(*pRoot)) < 0)
            *pRoot = RotateRR(*pRoot);
        else
            *pRoot = RotateRL(*pRoot);
    }

    return *pRoot;
}
```

균형 인수가 +2 이상이면 왼쪽으로 길게 불균형을 이룬 상태이므로 LL 또는 LR상태이다!

균형 인수가 -2 이하면 오른쪽으로 길게 불균형을 이룬 상태이므로 RR 또는 RL상태이다!

LL상태와 LR상태, 그리고 RR상태 RL상태의 구분



GetHeightDiff(GetLeftSubTree(*pRoot))의 반환 값이 0보다 크면 LL상태 그렇지 않으면 LR상태

GetHeightDiff(GetRightSubTree(*pRoot))의 반환 값이 0보다 작으면 RR상태 그렇지 않으면 RL상태

BinarySearchTree2.c의 실질적 변화

```
void BSTInsert(BTreeNode ** pRoot, BSTData data)
{
    . . . .
    Rebalance(pRoot);     // 노드 추가 후 리밸런싱!
}

BTreeNode * BSTRemove(BTreeNode ** pRoot, BSTData target)
{
    . . . .
    Rebalance(pRoot);     // 노드 제거 후 리밸런싱!
    return dNode;
}
```

앞서 소개한 함수의 확장 결과(예측)

실질적인 변화 결과

```
void BSTInsert(BTreeNode ** pRoot, BSTData data)
{
    . . . .
    *pRoot = Rebalance(pRoot);     // 노드 추가 후 리밸런싱!
}

BTreeNode * BSTRemove(BTreeNode ** pRoot, BSTData target)
{
    . . . .
    *pRoot = Rebalance(pRoot);     // 노드 제거 후 리밸런싱!
    return dNode;
}
```

AVL 트리의 동작결과 확인: main 함수

```
int main(void)
{
    BTreeNode * avlRoot;
    BTreeNode * clNode;        // current left node
    BTreeNode * crNode;        // current right node
    BSTMakeAndInit(&avlRoot);

    BSTInsert(&avlRoot, 1);
    BSTInsert(&avlRoot, 2);
    BSTInsert(&avlRoot, 3);
    BSTInsert(&avlRoot, 4);
    BSTInsert(&avlRoot, 5);
    BSTInsert(&avlRoot, 6);
    BSTInsert(&avlRoot, 7);
    BSTInsert(&avlRoot, 8);
    BSTInsert(&avlRoot, 9);

    printf("루트 노드: %d \n", GetData(avlRoot));
    clNode = GetLeftSubTree(avlRoot);
    crNode = GetRightSubTree(avlRoot);
    printf("왼쪽1: %d, 오른쪽1: %d \n", GetData(clNode), GetData(crNode));

    clNode = GetLeftSubTree(clNode);
    crNode = GetRightSubTree(crNode);
    printf("왼쪽2: %d, 오른쪽2: %d \n", GetData(clNode), GetData(crNode));

    clNode = GetLeftSubTree(clNode);
    crNode = GetRightSubTree(crNode);
    printf("왼쪽3: %d, 오른쪽3: %d \n", GetData(clNode), GetData(crNode));

    clNode = GetLeftSubTree(clNode);
    crNode = GetRightSubTree(crNode);
    printf("왼쪽4: %d, 오른쪽4: %d \n", GetData(clNode), GetData(crNode));
    return 0;
}
```

BinaryTree3.h
BinaryTree3.c
BinarySearchTree3.h
BinarySearchTree3.c
AVLRebalance.h
AVLRebalance.c
AVLTreeMain.c

파일구성

```
루트 노드: 5
왼쪽1: 4, 오른쪽1: 6
왼쪽2: 3, 오른쪽2: 7
왼쪽3: 2, 오른쪽3: 8
왼쪽4: 1, 오른쪽4: 9
```

실행결과

실행결과는 완전히 균형을 잡아 주지 못함을 보여준다!

수고하셨습니다~

Chapter 12에 대한 강의를 마칩니다!